IC Optimisation using Parallel Processing and Response Surface Methodology

Thesis submitted by

Godfrey Jonathan Gaston

for the degree of

Doctor of Philosophy

Edinburgh Microfabrication Facility
Department of Electrical Engineering
University of Edinburgh
Scotland

October 1992

Abstract

Simulation software has become an essential tool in the design and development of integrated circuits. The key to the rapid and efficient designs required in the competitive industry lies with the use of these simulators with statistical optimisation methods. This is necessary if companies are to survive in the aggressive IC marketplace. The linking of simulation and statistics not only results in rapid development times, but also in robust, highly manufacturable products.

In this work an automated software system is presented where the benefits of simulation and statistical techniques can be readily made available. The efficiency of the system is increased further by utilising parallel processing techniques. Using one system built around the INMOS transputer and one using Intel i860 processors, the time taken to obtain simulation results is greatly reduced.

Two statistical methods are investigated, namely Response Surface Methodology (RSM) and Taguchi analysis. In order to illustrate how these approaches can be utilised in the field of semiconductors, part of a 1.5μm nMOS process is optimised in relational to some specified device parameters. A comparison is made between both techniques, with good agreement being obtained.

Finally devices have also been fabricated using the same experimental design as for the RSM simulation analysis. This facilitated a verification of the simulation optimisation with reality. Both simulated and fabricated devices suggested the same improved optimised conditions when compared to the existing process parameters.

Acknowledgements

I wish to express my thanks to all those people who helped in any way towards this PhD. In particular, I would like to thank my supervisor, Dr A J Walton, for his constant help and guidance throughout the last three years.

A special word of thanks must also go to Mr Z Chen, whose advice, counsel and encouragement were never ending. I would also like to thank Dr W J C Alexander for answering so many of my pleas for help and to Dr L Clarke for helping me down the parallel processing road.

I am also indebted to the staff of the Edinburgh Microfabrication Facility (EMF) for all their help in fabricating the silicon wafers and to my fellow PhD students for their support and assistance in this work.

Last, and definitely by no means least, I wish to express my thanks to my parents whose inspiration and example have always been an encouragement to me, and to my wife, Julia, who laboriously typed this thesis and who was always there when things weren't going so well. Maybe I'm most indebted of all to my baby son Matthew, whose arrival ensured prompt completion of this thesis.

I conclude with an acknowledgement to the Department of Education for Northern Ireland, who funded this research.

Table of Contents

List of Figures

Chapter 1

Introduction

From the production of the first integrated circuit in 1959, the growth of the semiconductor industry has been staggering. This growth has been due to the ever increasing market for electronics products and also due to the tremendous advances and technological breakthroughs made in the design and fabrication of integrated circuits.

Miniaturisation breakthroughs have advanced at a steady pace. Every few years a new technology generation is announced with 0.5μm CMOS prototypes available at the end of 1992 and work is currently underway for design of sub 0.5μm CMOS devices.

One of the major driving forces behind these advances has been the Electronic Data Processing or EDP market. This has been dominated by the industry's insatiable hunger for larger amounts of compute power (in the form of the CPU) and for faster and higher density SRAM's and DRAM's. Figure 1–1 shows the trends in DRAM's in terms of size and cost.

There is emerging, however, another market in the semiconductor field, that of consumer electronics [1]. These applications are often more demanding in terms of cost, reliability, ease-of-use and power dissipation when compared to the EDP market.

One aspect of the market that requires attention is that of the portable consumer products, such as portable computers and camcorders. These products

Trends in DRAM development

	1985	1989	1993	1997	2000
DRAM size	256K	1M	4M	16M	64M
Wafer dia (in)	6	6	6	8	8
CD (microns)	1.25	1.0	0.7	0.5	0.3
Wafer/year (K)	100	80	70	50	20
Cost (M$)	100	250	720	1320	2500
Price waf (K$)	1.2	3.75	12.3	31.7	150
Yield (%)	90	85	80	70	70
Good die/waf	400	215	118	78	36
Cost/die ($)	3	17.5	105	406	4200
Cost/bit	1	1.75	2.6	2.5	6.5

Figure 1–1: Trends in fabrication of DRAM's

demand low power CMOS devices to preserve battery life. The challenge is to reduce the leakage of isolation and junctions. Two areas of particular importance in achieving this are firstly the use of rapid thermal annealing to activate dopant using a low thermal budget without the introduction of leakage, and secondly the lowering of the supply voltage [2].

One of the emerging technologies at present is that of BiCMOS [3]. These processes tend to be based around CMOS (due to low power considerations) but combine the superior drive capability of bipolar transistors. BiCMOS is particu-

larly useful in applications where a large gate drive is required to increase circuit fan-out or handle high capacitive loads. It is, therefore, likely that BiCMOS will become a more important player in the semiconductor industry in the years ahead, although CMOS is still likely to remain the workhorse for the foreseeable future.

1.1 Quality considerations in the IC industry

Early attempts at ensuring quality in manufacturing consisted of inspection of the final product before shipping. This was followed by introducing inspection earlier in the manufacturing process and, if possible, re-working product that fails to meet designated quality standards. The problem with these approaches is that unless something fundamental is changed in the manufacturing process, no improvement will ever be gained in product yield.

As the complexity of VLSI circuits increases and device dimensions continue to shrink, the sensitivity of IC fluctuations inherent in fabrication also increases. In order to achieve an acceptable or profitable yield, design methodologies were developed in order to survive in the aggressive semiconductor marketplace. One such methodology is Design for Manufacturability (DFM). This allows for variations in device fabrication parameters in the design phase, leading to increased yield, minimum time to market and hence high profit, in essence, pushing quality into the design stage, rather than relying purely on inspection processes.

Quality concerns are now at the forefront of many of the large semiconductor companies' objectives. Examples of these are the Total Quality Management programmes that are in force with companies such as Motorola [4] and Philips [5]. These procedures aim at six sigma quality, with an ultimate aim of zero defects. They also seek to comply to adopted quality standards such as ISO 9000.

1.2 Motivation

In order to implement useful practical approaches, the use of statistical optimisation methods and accurate, computationally efficient techniques for statistical modelling of IC processes have been developed. These techniques include Response Surface Methodology (RSM) and Taguchi methods.

Much of the previous work in this area has been in relation to real experimentation [6-8]. The purpose of this work was to look at the area of statistical device optimisation using simulators to model the fabricated devices. One of the problems with this approach is the often prohibitive amount of CPU time required to perform all the required simulations. Previous work that utilised simulators in the optimisation process have commented on the problem with excessive CPU time [9, 10]. To overcome this problem, parallel processing techniques were implemented to reduce CPU time. A comparison of RSM and Taguchi techniques provided a useful insight into the relative merit of both techniques.

1.3 Thesis outline

This chapter has given a brief overview of the trends in the semiconductor industry and the relevance of this field of study to the wider concept of statistical optimisation and DFM.

Chapter 2 outlines the general concept behind process simulators and their use in the IC industry. A software review details some of the packages currently available on the market. The doping profiles from process simulators can be loaded into device simulation programs for device analysis. This is described in **chapter 3**, along with typical characteristics that can be obtained from such programs. Also outlined is the use of parameter extraction programs and integrated CAD tools in device characterisation.

Chapter 4 describes the theory behind the concept of design of experiments and details the RSM and Taguchi techniques. The RS/1 statistical package is also illustrated using a simple example from the field of microelectronics. The classification and development of parallel computing are given in **chapter 5** as well as a detailed description of the transputer and Intel i860 processor.

Chapter 6 describes the evolution of the automated software system and the different timing results obtained from the different sequential and parallel platforms utilised.

The results obtained for simulation of the EMF 1.5μm nMOS process and subsequent optimisation of part of the process is given in **chapter 7**. Also outlined is a comparison between RSM and Taguchi optimisation techniques. These results are compared with reality in **chapter 8** where two types of statistical design are also contrasted.

Finally in **chapter 9** some brief conclusions are drawn up together with some ideas for future work.

Chapter 2

Process Simulation

2.1 Introduction

The use of process simulators as a tool for semiconductor fabrication began in the late 60's and early 70's. Such simulators were very basic in nature and utilised simple analytical models [11]. As fabrication technology increased in complexity and sophistication, so these original models became more inaccurate. As a result in 1977 the SUPREM I simulation program was written at Stanford University [12], this then led to SUPREM II [13]. This was capable of numerically simulating in one dimension most of the steps in the integrated circuit (IC) fabrication process technology of that period. Development of one-dimensional simulators continued and this resulted in SUPREM III [14]. This program uses sophisticated models for the fabrication process and also allows more layers to be specified thus leading to more reliable results with a minimum of computation time. The one-dimensional doping profiles from SUPREM III can be input into a two-dimensional device simulator, when allowances are made for two-dimensional effects.

As device geometries decreased it became evident that two-dimensional simulators were required to accurately simulate short channel effects. Device characteristics such as punchthrough voltage are strongly related to lateral diffusion and such effects can only be simulated properly in two dimensions. These two-dimensional simulators, such as SUPRA [15], were also extremely useful in modelling oxide formation over different geometries.

One of the purposes of process simulation is to produce a doping profile and the structure to be used in a device simulation program, ultimately leading to a

set of I-V characteristics. Simulations also have a secondary effect of providing more insight into the device than could be obtained from the traditional experimental approach. This can include information such as depletion regions and concentration contours.

One of the most up to date process simulation packages is SSUPREM4, [16], a fully numerical two-dimensional package. This utilises among other things, advanced point defect models for impurity diffusion and new two dimensional oxidation models. The models are based on large amounts of experimental data obtained from oxidation and impurity profiles extracted from test structures. The benefits of these physical models are that the simulator requires less calibration than those programs with simpler models. Thus the results obtained are less dependent on the fabrication line in question. It is, however, often necessary to carry out some calibration to obtain more accurate results. Calibration usually entails taking known values from the fabrication line and comparing them to those obtained from simulation. An example of one of these values is oxide thickness. In MOS devices the gate oxide thickness is a very important parameter and it is essential to obtain agreement between simulation and reality.

2.2 Operation of SSUPREM4

The following sections describe the salient features of SSUPREM4 by considering the input simulation file, listed below.

```
#define some parameters
set echo
option quiet

#the x dimension definition
line x loc = 0.0 tag = left
line x loc = 0.25 spacing = 0.25 tag = right

#the y dimension definition
line y loc = 0 spacing = 0.02 tag = top
```

```
line y loc = 0.50 spacing = 0.02
line y loc = 2.0 spacing = 0.25 tag=bottom

#the silicon wafer
region silicon xlo = left xhi = right ylo = top yhi = bottom

#define the exposed surfaces
bound exposed xlo = left xhi = right ylo = top yhi = top

#initialise the mesh
init boron conc=1.0×10^14

#phosphorus source/drain implant
implant phosphorus dose=3×10^15 energy=70 pearson

#diffusion step
diffuse time=30 ramp start=1000 stop=1150 dryo2

# deposit and etch
deposit aluminum thick=1
.
.
etch aluminum all

#save the data
structure pisces=pisces.str

quit
```

Each of the steps outlined above will now be considered in more detail.

2.2.1 Grid Definition

The following section describes the use of the grid in SSUPREM4 and outlines the significance of the choice of grid in obtaining accurate solutions. Also described is the method of initialising the structure.

```
#define some parameters
set echo
option quiet

#the x dimension definition
line x loc = 0.0 tag = left
line x loc = 0.25 spacing = 0.25 tag = right

#the y dimension definition
line y loc = 0 spacing = 0.02 tag = top
line y loc = 0.50 spacing = 0.02
line y loc = 2.0 spacing = 0.25 tag=bottom
```

In order to simulate the various fabrication steps, SSUPREM4 expresses the wafer as a two-dimensional structure. This structure is divided into a mesh or grid. The above lines containing the statements `line x` and `line y` are used to specify the horizontal and vertical locations of the grid lines. The `spacing` term, as the name suggests, determines the distance between adjacent grid lines. This grid consists of a series of triangles connected together. Figure 2–1 shows the grid generated by the above statements. There are only two `line x` statements, with the spacing equal to the distance between them. This results in only two lines in the vertical direction. Since the spacing for the y direction is small, this results in a number of lines in the horizontal direction. An example of a grid used to analyse a small geometry MOS process is illustrated in figure 2–2.

It can be seen from figure 2–2 that the grid is finer in the channel region (to the left of the figure) and less dense in the field region (to the right) and deep into the substrate. This is because it is in the channel region that the most accurate results are required. It should be noted that this is only half of a real device and before this structure could be used for device analysis it would have to be reflected about the line x=0. This technique of reflection for symmetrical devices is useful in the sense that it speeds up the simulation as only half the number of nodes need to be simulated.

The points where the solution is evaluated are called nodes. At each node the

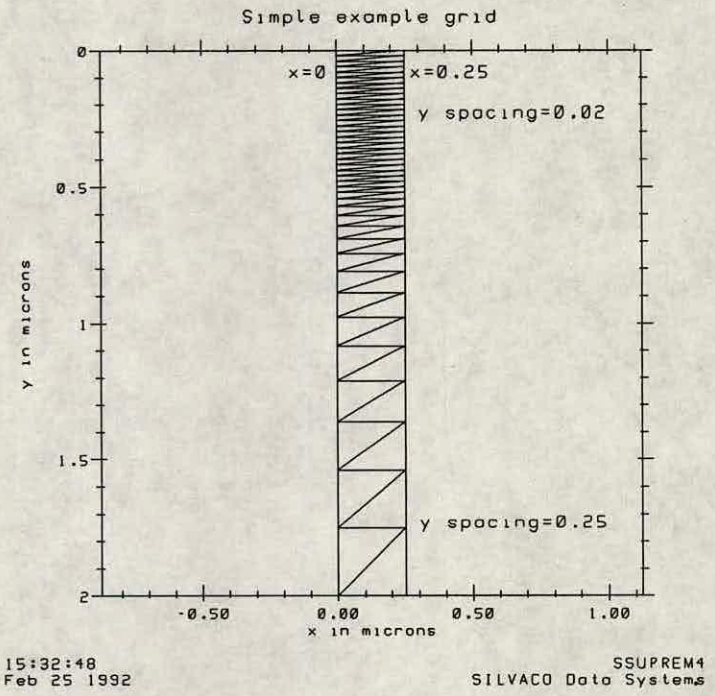


Figure 2–1: An example of a simple grid generated by SSUPREM4

solution is obtained using finite element analysis. There is a trade-off between choosing too fine a grid, which results in long simulation times and too coarse a grid, which leads to inaccurate results. The choice of the initial grid can have a significant bearing on the final results obtained. It is preferable to have a finer grid in the regions being investigated eg. depletion regions. Obviously a fine grid is not required deep down into the substrate. The choice of grid can often involve an iterative process of refinement, until the optimum grid is achieved. Figure 2–3 shows the channel region of a 1.5μm device using a coarse grid. Figure 2–4, describes the same device except using a finer grid. The finer grid obviously results in a much superior doping profile, with the coarse grid likely to lead to erroneous device characteristics eg punchthrough voltage.

The problem of analysis using a grid is particularly complicated in process simulation due to the problem of moving boundaries during oxidation steps. This problem does not occur in device simulation as boundaries are fixed.

The portion of the data file below sets up the region and physical boundaries of the silicon wafer, indicating exposed surfaces. The `init` command calculates the

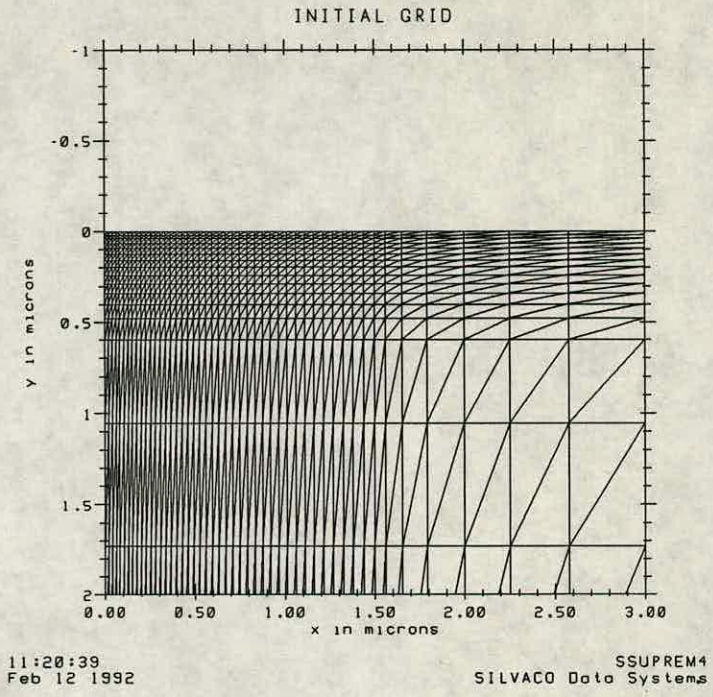


Figure 2–2: An example of a SSUPREM4 grid used to simulate a small geometry process

grid or mesh from the `line x` and `line y` statements and sets the initial substrate concentration of 1.0×10^{14} atoms cm^{-3}.

```
#the silicon wafer
region silicon xlo = left xhi = right ylo = top yhi = bottom

#define the exposed surfaces
bound exposed xlo = left xhi = right ylo = top yhi = top

#calculate the mesh
init boron conc=1.0×10^14
```

2.2.2 Implantation

In SSUPREM4 implantation is performed by specifying the `implant` statement. There is a choice of `antimony`, `arsenic`, `boron` and `phosphorus` impurities, with `dose` specifying the number of impurity atoms cm^{-2} and `energy` indicating the

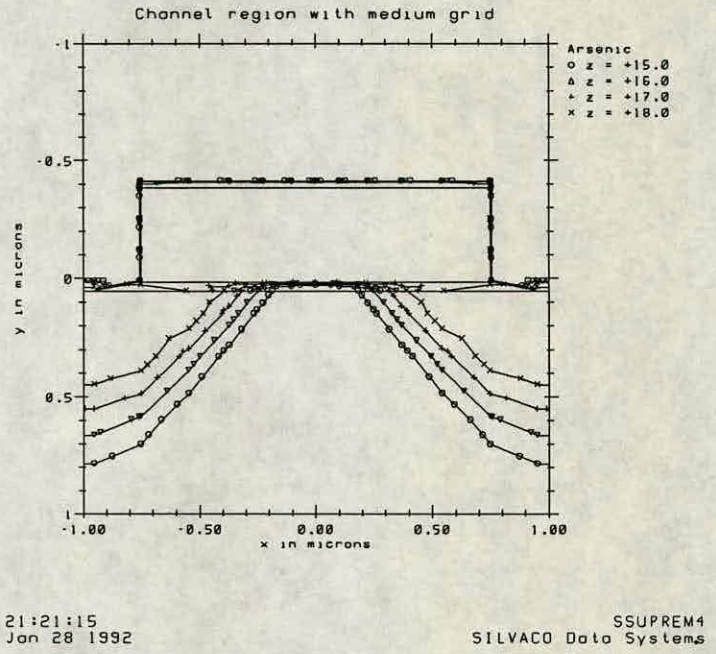


Figure 2–3: Channel region using a coarse grid

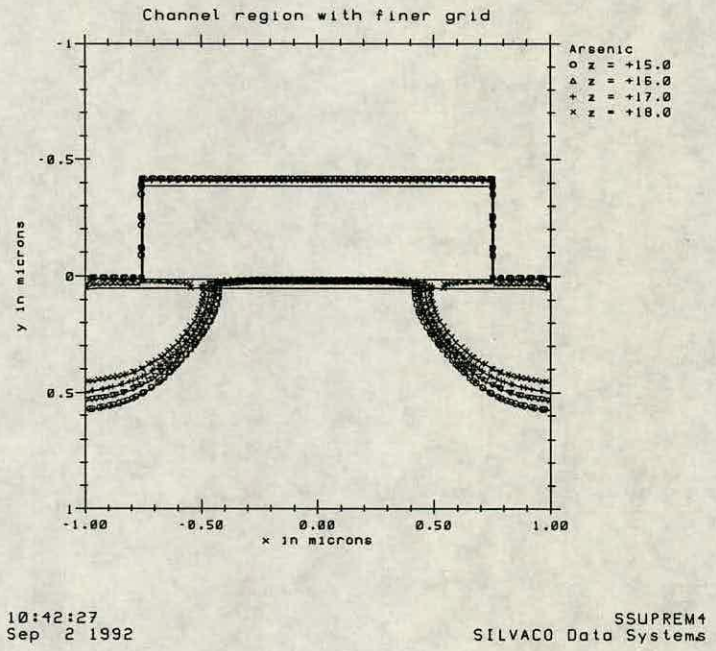


Figure 2–4: Channel region using a finer grid

implant energy in keV. There are three different types of models available; a simple Gaussian distribution, a Pearson IV distribution and a dual Pearson distribution.

Figure 2–5 shows a comparison of a Gaussian profile and a dual Pearson profile for an arsenic implant. The pointed nature of the profiles is due to the low number of grid points for this simulation. For serious simulations more grid points would be used, thus ensuring smooth profiles. The boron concentration is shown to give an indication of the junction depth. Figure 2–6 shows a concentration contour plot for the source drain implant of a MOS transistor, after diffusion steps.

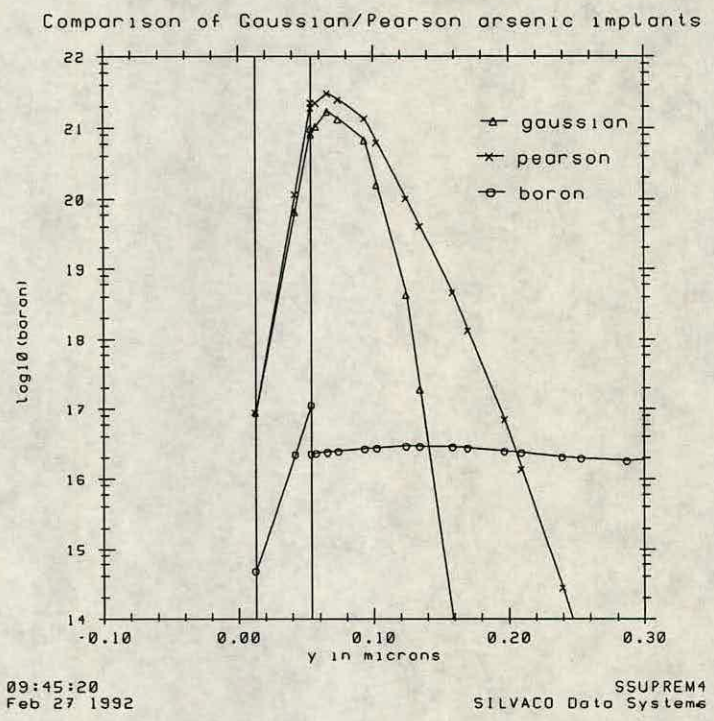


Figure 2–5: A comparison of Gaussian and Pearson distributions for an arsenic implant

The following gives an example of a phosphorus implant with a dose of 3×10^{15}, an energy of $70keV$ and using the dual Pearson distribution.

```
#phosphorus source/drain implant
implant phosphorus dose=3×10^15 energy=70 pearson
```

2.2.3 Diffusion and Oxidation

Diffusion is the basic process used in SSUPREM4 and is performed using the `diffuse` statement. Any impurities present are diffused and if oxygen is present,

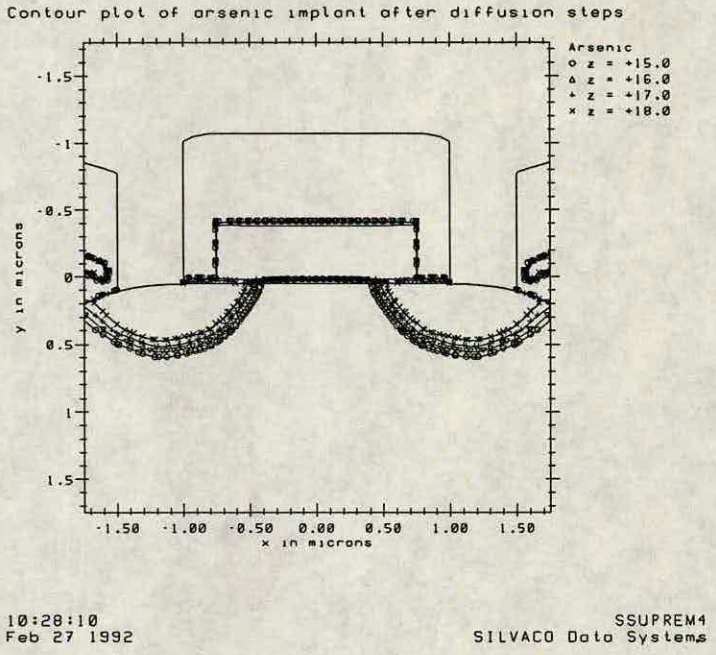


Figure 2–6: Concentration contour plot for the source drain implant of a MOS transistor

oxidation is also performed. The time spent in the furnace is specified by the `time` command and the furnace temperature by the `temp` command. Oxidation can be carried out by specifying either `dryo2` or `weto2`, during diffusion.

An example dry oxidation is outlined below.

```
#diffusion step
diffuse time=30 ramp start=1000 stop=1150 dryo2
```

Diffusion is divided into two areas, namely:

- Diffusion of impurities
- Diffusion of point-defects

The former, as the name suggests, is the mechanism whereby impurities undergo diffusion in the substrate. The latter is concerned with the diffusion of interstitials and vacancies in the silicon.

The movement of dopants by vacancy and interstitials (interstitialcy) are of major importance for the diffusion of boron, phosphorus, arsenic and antimony. It is generally considered that at low temperatures and under non-oxidising conditions, diffusion is primarily via vacancies. Oxidation, however, injects interstitials into the bulk and in the case of boron, phosphorus and arsenic, results in oxidation enhanced diffusion and oxidation retarded diffusion in the case of antimony.

It is important when considering diffusion to take into account the diffusion coefficients for the implanted element. Figures 2–7 and 2–8 show how calibration of coefficients is necessary for proper simulation and operation of devices. As before the pointed nature of the profiles in these two figures is due to the low number of grid points generated. Figure 2–7 shows how phosphorus, when implanted for source and drain regions and then annealed, diffuses through the polysilicon gate and gate oxide. Thus there is an unwanted high concentration of phosphorus in the channel area. By changing the diffusion coefficient of phosphorus in oxide the phosphorus no longer diffuses through to the channel area. This can be seen in figure 2–8.

The `oxide` statement can be used to specify all the parameters relating to oxidation, including the type of model to be used. A present limitation of SSUPREM4 is that oxidation cannot be implemented at a point where oxide, silicon and a third material (eg. nitride) meet.

There are five models supported by SSUPREM4, all of which have their origins with the Deal and Grove model [11], which solves the one dimensional growth rate equation for constants A and B. These constants are used in the different SUPREM4 models. These models are as follows:

1. ERFC

2. ERFG

3. Vertical

4. Compress

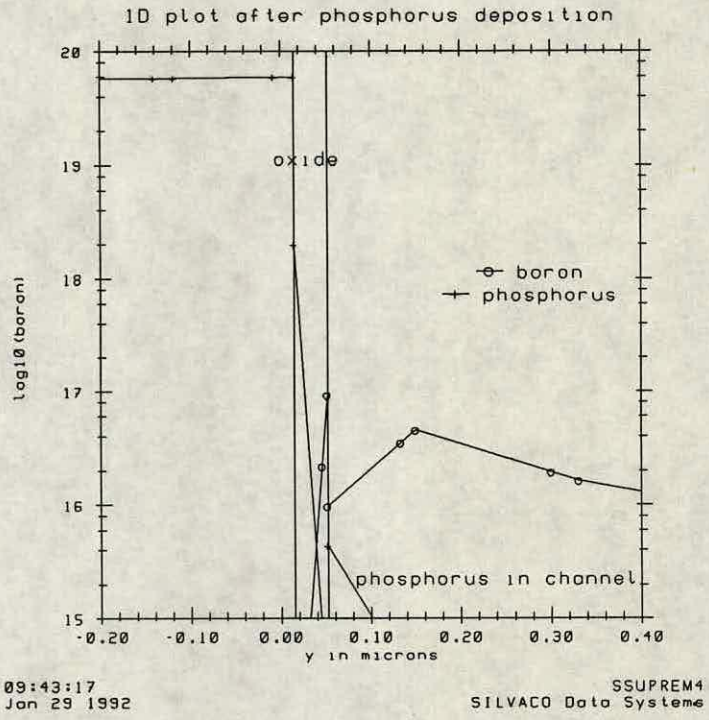


Figure 2–7: Phosphorus diffusion using default coefficients

5. Viscous

These five models can be broken down into 2 basic sections:

1. Analytical models

2. Numerical models

Analytical Models Included in this section are the ERFC and ERFG models. ERFC is the simplest model used in SSUPREM4 and describes the bird's beak growth of oxide. ERFG is a more complex model and allows for oxidation under a nitride mask. This model includes two others, namely ERF1 and ERF2. The former is for use with thin nitride masks and the latter for thick nitride masks.

Numerical models The remaining three models, vertical, compress and viscous come under this category. These models are necessary for the accurate simulation of more complex structures used in more modern processes. They directly solve

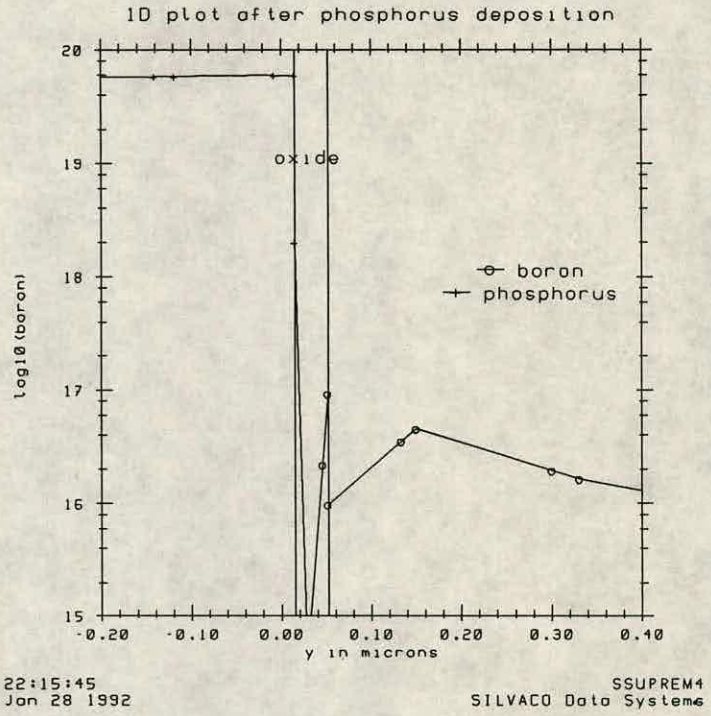


Figure 2–8: Phosphorus diffusion using calibrated coefficients

the models proposed by Deal and Grove [11], taking into account the oxide flow due to expansion during the oxidation process.

The vertical model is the simplest numerical model. As the name suggests, oxide growth is purely in a vertical direction and is not suitable for trenches and non planar structures.

The compress model calculates the oxide growth in two dimensions using the finite element method. It includes, however, a small measure of compressibility and thus cannot calculate stress accurately.

The viscous model is similar to the compress model except that stress analysis is calculated much more accurately, since no compression is taken into account.

2.2.4 Deposition and Etching

Unlike the other processes described above, deposition and etching are not solved using physical models. They are simulated at low temperatures and thus diffusion

of any impurities can be ignored. The simulation is considered as a geometric problem and is solved by defining specific regions of interest.

Deposition is simulated using the `deposit` statement. This provides the facility to define the material to be deposited, the deposit thickness, the type of doping in the deposited layer, the concentration of this doping etc. To obtain more accurate results it is often necessary to specify the number of divisions in the grid (of the deposited layer) to be greater than the default setting of one.

Etching is carried out using the `etch` command and can be used to etch materials in different layers in various regions or sections of the structure. Specific thicknesses can be removed and values for etch defined. For more complicated structures it is advisable to specify a grid line exactly at the edges of the etched part of a deposited layer. This helps ensure that invalid grids are not generated.

```
# deposit and etch
deposit aluminum thick=1 divisions=3
.
.
etch aluminum all
```

2.2.5 SSUPREM4 Interface

SSUPREM4 can be interfaced with both the one dimensional process simulator, SSUPREM3 and the two dimensional device simulator S-PISCES 2B [17]. The SSUPREM4 `profile` statement can read in a one dimensional impurity profile generated by SSUPREM3.

More importantly for the purposes of this work, is the interface with S-PISCES 2B. In order to save the structure file in SSUPREM4 in a form suitable for PISCES the "`structure pisces=`" command is used as shown below. The method for loading this file into S-PISCES 2B will be dealt with in section 3.2.1.

```
#save the data
structure pisces=pisces.str
```

```
quit
```

The ability to load process simulation structures into device simulation programs, has heralded the use of such programs in an integrated CAD framework.

2.3 Software Review

There is at present a large diversity of different process simulation programs available on the market. Apart from SSUPREM4, SUPRA and SUPREM III, (which are outlined in section 2.1) other one dimensional and two dimensional programs have been described in the literature.

One of the earliest one-dimensional process programs to be developed was ICECREM [18]. This program has the capability to simulate ion implantation, diffusion steps, etching and deposition of oxide and nitride. One of the major attractions of this program is the ease of use for those inexperienced in the realm of process simulation. As a consequence reasonably accurate results can be obtained in a very short period of time, due to the simple input syntax and low computational requirements.

As device geometries began to shrink it was essential to accurately simulate the lateral diffusion of small channel devices. This led to the development of LADIS [19]. LADIS sought to produce precise doping profiles after ion-implantation and drive-in anneals. These two-dimensional profiles were then able to be input into device programs.

A common two-dimensional process simulator in use at present, is COMPOSITE [20]. This was one of the first simulators to include the capability to simulate all of the important semiconductor fabrication steps. As well as the usual diffusion, oxidation and ion-implantation steps, also included is lithography, etching and deposition. Its modular structure facilitates upgrading and is attractive to those learning simulation, due to its simple menu driven nature.

Another two dimensional process simulator available at present is TITAN [21]. This was developed at Grenoble, France and provides the capability of performing the standard technological steps eg implantation, diffusion, etc, as well as more advanced plasma-based process steps. These steps include plasma enhanced chemical vapour deposition, sputter etching and reactive ion etching. It is linked with a device simulator to enable fast and efficient realisation of device characteristics.

Other process simulators which have been developed include RECIPE [22], FEDSS [23], BICEPS [24] and PREDICT [25].

Chapter 3

Device Simulation and Parameter Extraction

3.1 Device Simulation

3.1.1 Introduction

In the real world of semiconductor fabrication, the developing of new processes is closely linked with device characteristics. Thus, if a device does not operate as required, then the process will have to be adjusted, more wafers fabricated and devices retested. As one can imagine this is often a time consuming and laborious process. This provided the stimulus for the development of device simulation programs and their integration with process simulators.

The origins of semiconductor device analysis, using numerical, instead of analytical techniques, can be traced to Gummel [26] and the work he carried out. The general principle behind device analysis lies in the solution of a set of nonlinear partial differential equations governing the semiconductor device operation. These equations are often referred to as the semiconductor equations. They consist of Poisson's equation and both electron and hole current continuity equations. They will be outlined in detail in section 3.1.2.

Some of the more popular device simulation packages available today are SPISCES-2B, [17], MINIMOS, [27] and SEDAN-3 [28]. PISCES-2B is capable of modelling two dimensional distributions of carrier and potential concentrations.

From such information, electrical characteristics can be obtained for a large number of different devices, namely MOS and bipolar transistors and diodes. It operates in a similar manner to SSUPREM4, in that it divides the structure into a grid and solves the equations at each node.

Just as the shrinking of device dimensions resulted in the evolution of two dimensional process simulators from one dimensional simulators, such dimensions have led to three dimensional device simulation programs [29-32]. These programs are necessary for accurate simulation of narrow channel devices. Some of the major effects of narrow channels are shift of threshold voltage and change in conductivity. These programs will be used even more in the future as device geometries shrink yet further.

3.1.2 Semiconductor Equations

These equations govern the electrical behaviour of semiconductor devices under the influence of an electric field. They consist of Poisson's equation and the electron and hole continuity equations and are outlined in this order below.

$$\nabla \cdot (\epsilon \nabla \phi) = -q(p - n + N) \tag{1.1}$$

$$\nabla \cdot J_n = qU(n,p) + q\frac{\partial p}{\partial t} \tag{1.2}$$

$$\nabla \cdot J_p = qU(n,p) + q\frac{\partial p}{\partial t} \tag{1.3}$$

Where ϵ is the dielectric permittivity, ϕ is the electric potential, q is the electronic charge, p and n are the hole and electron concentrations, N is the net impurity concentration, J_p and J_n are the hole and current densities and U is the net recombination rate.

These equations are discretized on a simulation grid resulting in a set of algebraic equations which are then solved by non linear iterative methods. A more detailed analysis of these equations can be found in [33].

3.2 Operation of SPISCES-2B

As was the case for SSUPREM4, the operation of SPISCES-2B will be described by considering a simple input simulation file. Although this does not include all the options available with such a complex device simulation program, it illustrates the important principles involved.

```
$ specify mesh
mesh rectang nx=15 ny=15 outfile=p1_mesh smooth=1
x.mesh n=1 l=-1.5
x.mesh n=15 l=1.5

y.mesh n=1 l=-1.2
y.mesh n=4 l=0
y.mesh n=10 l=0.4
y.mesh n=15 l=2

$ specify device for plotting
option term=ps plot.file="plot.ps"

$ input doping profile from ssuprem4
doping suprem4 infile=comp.str

region num=1 silicon ix.low=0 iy.hi=2
region num=2 oxide ix.hi=3 iy.hi=-0.5
region num=3 polysili ix.low=0 iy.low=-0.1
region num=4 oxide ix.low=0 iy.low=-0.6
region num=5 oxide ix.low=-3 iy.hi=-0.5
region num=6 oxide ix.low=0 iy.hi=-1.18

$ electrodes:
electr num= 1 x.min=-0.750 x.max= 0.750 y.min= -0.382 y.max= 0.019
electr num=2 x.min= 1.000 x.max= 1.500 y.min= -1.005 y.max= 0.100
electr num=3 x.min=-1.500 x.max= -1.000 y.min= -1.005 y.max= 0.100
electr num=4 x.min=-3.000 x.max= 3.000 y.min=2.000 y.max=2.000
```

```
plot.2d grid title="initial grid"

contact num=3 n.poly

regrid doping ratio=2 log smooth=1
plot.2d grid title="1st doping regrid"

regrid doping ratio=2 log smooth=1
plot.2d grid title="2nd doping regrid"

$ specify physical models to use
models conmob fldmob surfmob
symb carriers=0
method iccg damped
solve init

symb carriers=0
solve v1=8 local

$Use Gummel's method for the solution
symb newton carriers=2

$Setup log file for IV data
log outf=punch_log
solve v1=8 elec=1 vstep=0.1 nstep=20
```

The above steps will now be described in more detail.

3.2.1 Input from process simulation

Just as device operation or analysis is closely linked to process parameters in actual fabrication, so device simulation is closely linked to process simulation in the realm of modelling. As progress was made in the operation and accuracy of process simulation programs such as SSUPREM4, this in turn led to improvement and more accurate results from device programs. This is due to the fact that the doping profile from process simulators is used as an input to device simulators. As a result, if the doping profile is made more accurate, then the results from device

simulation will be more reliable. On the other hand it is futile to have accurate results from process simulations and then input these results into an unsatisfactory device program.

Taking SSUPREM4 and SPISCES-2B as an example, the following outlines how the profile can be loaded from SSUPREM4 into SPISCES.

A structure file, containing the doping profile information is created in S-SUPREM4, as outlined in section 2.2.5. This file can then be input into SPISCES in two different ways:

1. Reading in the SSUPREM4 doping profile and mesh

2. Reading in the SSUPREM4 doping profile only

Option 2 is generally a more appropriate option, as the mesh in SPISCES can then be specified and changed to suit the device simulation. Also with option 2, if points on the SPISCES mesh do not correspond with points on the SSUPREM4 mesh, then interpolation can be used to find the doping concentration. With option 1 the mesh in SPISCES is restricted to the one defined in SSUPREM4 and a good mesh for process simulation does not necessarily mean an adequate mesh for device simulation.

It is option 2 that is defined in the included example file.

```
doping suprem4 inf=comp.str
```

It is, however, necessary to specify the SPISCES-2B mesh using the `mesh`, `x.mesh` and `y.mesh` statements. The position of the electrodes must also be specified using the `electrode` statement, as well as the different regions in the structure using the `region` statement.

3.2.2 Refining the Mesh

For accurate device simulation it is often necessary to refine the mesh so that there is a fine mesh in the area of investigation and a coarse mesh in the area not under

consideration. The refinement process is carried out using the `regrid` command. The regrid works on the principle that if the specified criterion, eg doping, varies by greater than a certain specified ratio between two nodes, then the grid is made finer automatically at those nodes. In the above example file, successive regrids were carried out for doping and then for potential. The following lines specify the syntax for regridding and the necessary `plot.2d` statements for plotting out each regrid.

```
regrid doping ratio=2 log smooth=1
plot.2d grid title="1st doping regrid"
```

Figure 3–1, 3–2 and 3–3 show how the grid changes with the refinement process. The advantage of such a command is that it is carried out automatically and no detailed knowledge is required by the user.

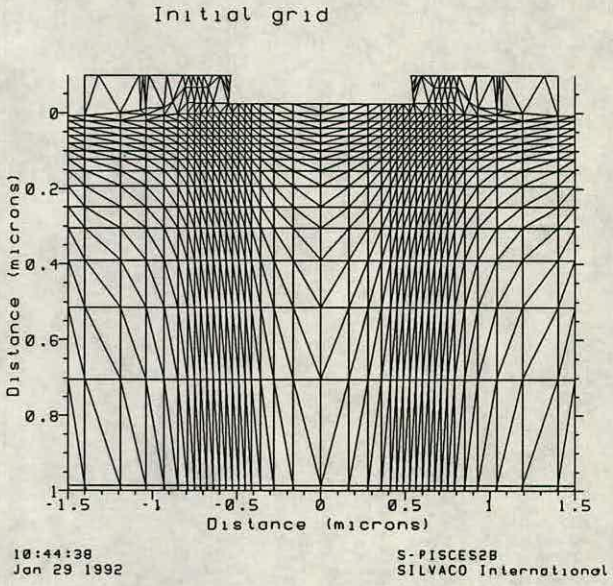


Figure 3–1: Initial grid definition

The `ratio = 2` parameter simply specifies the threshold value for regridding to take place, ie if the logarithm of the value of doping changes by 2 or more across a triangle, then that triangle is refined.

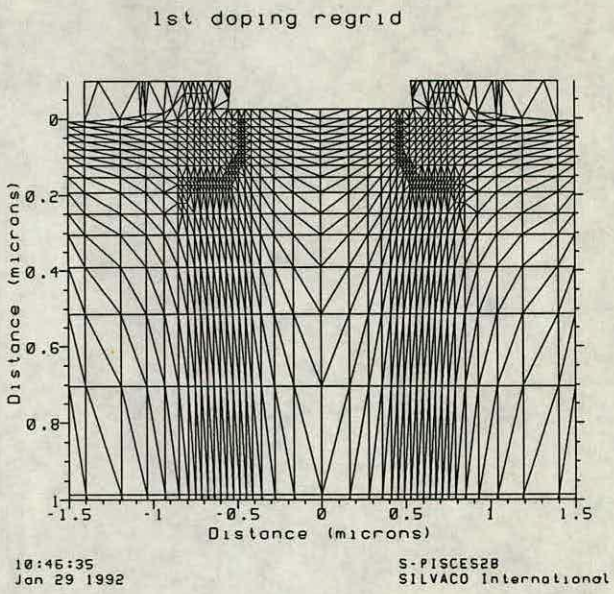


Figure 3–2: First doping regrid

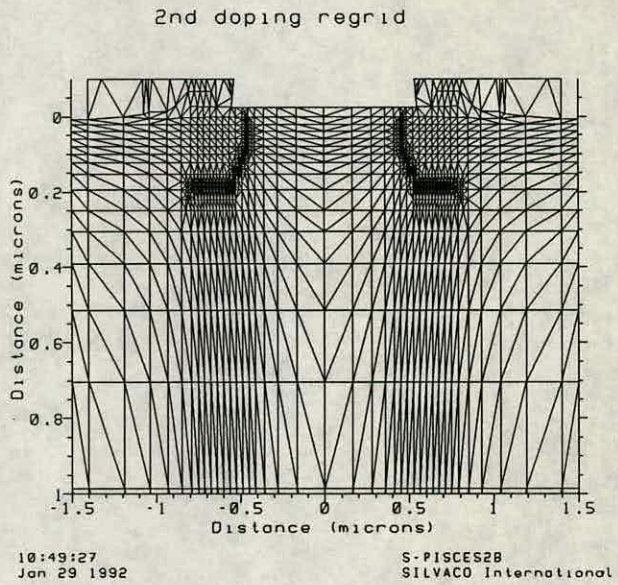


Figure 3–3: Second doping regrid

3.2.3 Solution Methods

The set of nonlinear algebraic equations formed on discretization of the grid are the subject of the different solution methods in SPISCES-2B. There are two basic types of solution methods which are used to solve the equations, namely Gummel's

method and Newton's method. Depending on the range of conditions, one method may be more appropriate than the other. In general, Newton's method is more stable, however, it is also slower from a computation point of view and also requires large amounts of memory.

The following outlines two different solution statements used in the example data file.

```
$ specify physical models to use
models conmob fldmob surfmob
symb carriers=0
method iccg damped
solve init

symb carriers=0
solve v1=8 local

$Use Gummel's method for the solution
symb newton carriers=2

$Setup log file for IV data
log outf=punch_log
solve v1=8 elec=1 vstep=0.1 nstep=20
```

Since the first solution method is only for an initial solution at zero bias, then neither Newton nor Gummel is specified on the `symb` line, `carriers = 0` simply means that a Poisson only solution is adequate. In the second solution method, since a bias is about to be applied, Newton is specified on the `symb` line, together with the number of carriers and type of carrier. The `method` statement sets up parameters associated with the solution algorithm chosen. Some method parameters are particular to either Newton or Gummel solutions and some are common to all solutions.

In incremental bias solutions it is important to choose an appropriate value for the step increase in voltage. If the chosen value is too large, then the program can have difficulty converging and can overflow and stop. If the value chosen is too small, then the time taken for solution can increase quite significantly. Thus it is

important to select a value which does not result in overflow, but also gives an acceptable computation time. To overcome the problem of large bias steps, one can use the `damping` command. This smoothes out the effects of the large bias steps for Gummel solutions.

3.2.4 Device Characteristics

The ultimate aim of device simulation is often to obtain typical device characteristics for the device in question. From these characteristics much information can be extracted which can be used to compare one device against another. Two of the most common characteristics used are, sets of drain current, I_d, versus gate voltage, V_{gs} for different values of substrate bias, V_{bs} and sets of I_d versus drain voltage, V_{ds}, for different values of V_{gs}. The former is plotted in figure 3–4 and the latter in figure 3–5.

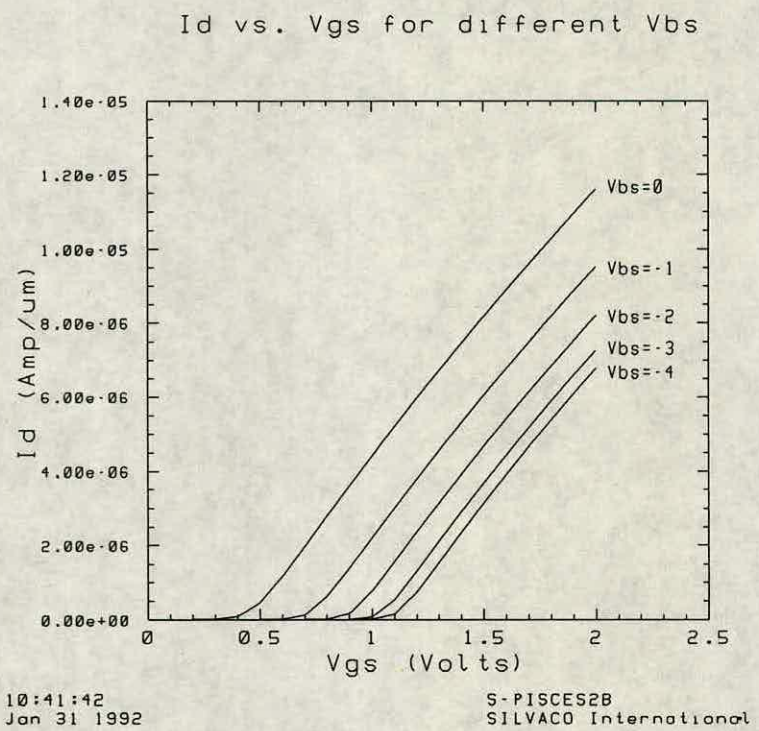


Figure 3–4: I_d versus V_{gs} for different V_{bs}

A typical file which creates the characteristics of figure 3–5 is outlined below.

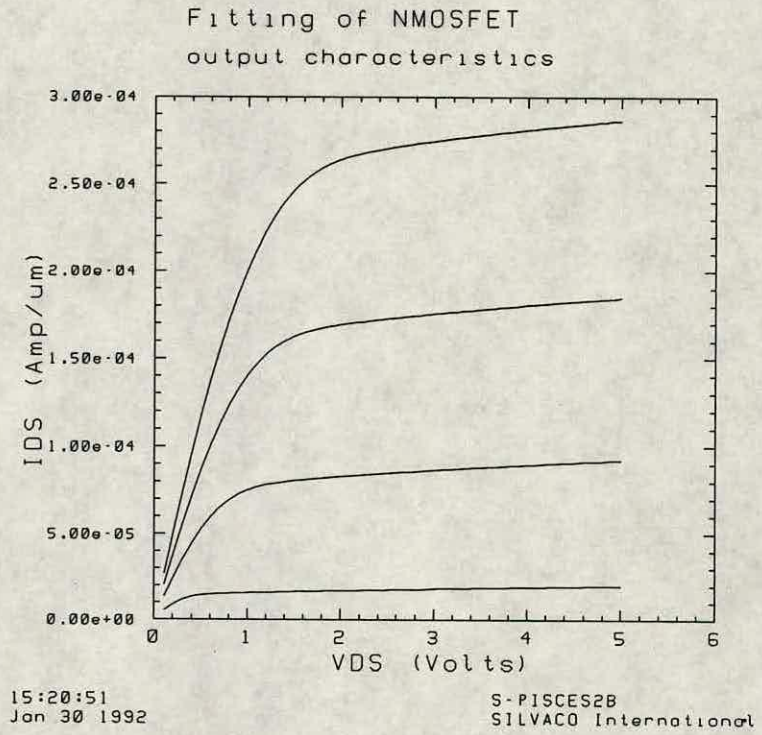


Figure 3–5: I_d versus V_{ds} for different V_{gs}

```
Title file:  nmos/plot.IdVd N-channel MOSFET example
$ N-channel MOSFET example
$ Plot-file:  Output characteristics
mesh inf=gmesh3
plot.1d inf=d411 x.ax=v4 y.ax=i4 abs x.label=VDS y.label=IDS
+ title="Fitting of NMOSFET"
+ subtitle="output characteristics"
plot.1d inf=d311 x.ax=v4 y.ax=i4 abs unch
plot.1d inf=d211 x.ax=v4 y.ax=i4 abs unch
plot.1d inf=d111 x.ax=v4 y.ax=i4 abs unch

end
```

The `inf` in the `mesh` statement loads in a previously saved mesh and the `inf` in the `plot.1d` statements loads in previously saved solution values. These values are then plotted to the specified output device.

A useful feature of SPISCES-2B is the `fit` statement. This can be used to

fit I-V curves that have been generated by the `plot.1d` statements. There are three different technologies which the command can fit, namely diode, bipolar and MOS. For the purpose of this study only MOS will be considered.

From a set of I_d versus V_{gs} curves it is possible to extract `VTO`, the threshold voltage, `NSUB`, the average doping concentration, `GAMMA`, the body effect coefficient and `PHI`, the built-in junction voltage.

From a set of I_d versus V_{ds} curves it is possible to extract values for `GDS`, maximum value of conductance, `ISAT`, maximum value of drain current and `RLIN`, parasitic resistance in the linear region.

3.3 Parameter Extraction

Since the devices modelled using process and device simulation programs will ultimately be used in circuits, the next obvious step in the simulation procedure is circuit simulation, the most common being SPICE [34]. In these programs are analytical device models, with the important criteria being their parameter values. Thus for accurate circuit simulation it is essential that these parameter values are also accurate. It is in the establishment of these values that parameter extraction programs are used. The input to parameter extraction programs can be either measured device characteristics or in this case, results from a device simulator.

According to the review by Wang *et al* there are three basic types of methods used in parameter extraction programs [35].

1. Graphical Methods
2. Analytical Methods
3. Numerical Methods

Graphical Methods This method involves the fitting of a set of device characteristics using graphical drawings.

Analytical Methods This involves the solution of a set of device analytical formulas obtained from device models for the extraction of the required parameters.

Numerical Methods These methods are the most popular in use at present and are used in the commercial package TOPEX [36]. They work on the principle of extracting the model parameters from a set of device characteristics that produce a least squares fit [37]. TOPEX uses a modified Levenberg-Marquardt algorithm which is reliable and usually convergent [38, 39].

3.3.1 Operation of TOPEX

Topex can be used to extract SPICE parameters for a wide variety of devices, namely bipolar, MOSFET, JFET and for junction capacitances. Since the subject of this work is for MOS devices, the MOS/SPICE model will be briefly outlined below. This model can extract a variety of parameters including, `VTO` the zero bias threshold voltage, `gamma`, the body effect coefficient, `NSUB`, the substrate doping and μ_0, the mobility.

Care must be taken, however, in the extraction of parameters from device characteristics and the user must be aware of "sensible" values for some device parameters. It is very possible to obtain quite different values of extracted parameters from the same set of device characteristics. This can be seen from figure 3–6 and figure 3–7 which are examples of how Topex fits curves to data points.

Figure 3–6 shows a fit with an error of 0.64% and a value of gamma of 0.7605 and a mobility of 842 $cm^2V - s^{-1}$. This would appear to have a better fit than that of figure 1–7, which has an error of 1.00%. On closer inspection, however, Ld (the lateral diffusion) for figure 1–6 is zero. It is known from practice that Ld should have a value of between 0.25 μm and 0.27 μm. Thus figure 1–7 represents a fit where Ld is allowed to vary within this range, thus giving a more realistic value for gamma (0.8274) and mobility (561), despite a seemingly "poorer" fit.

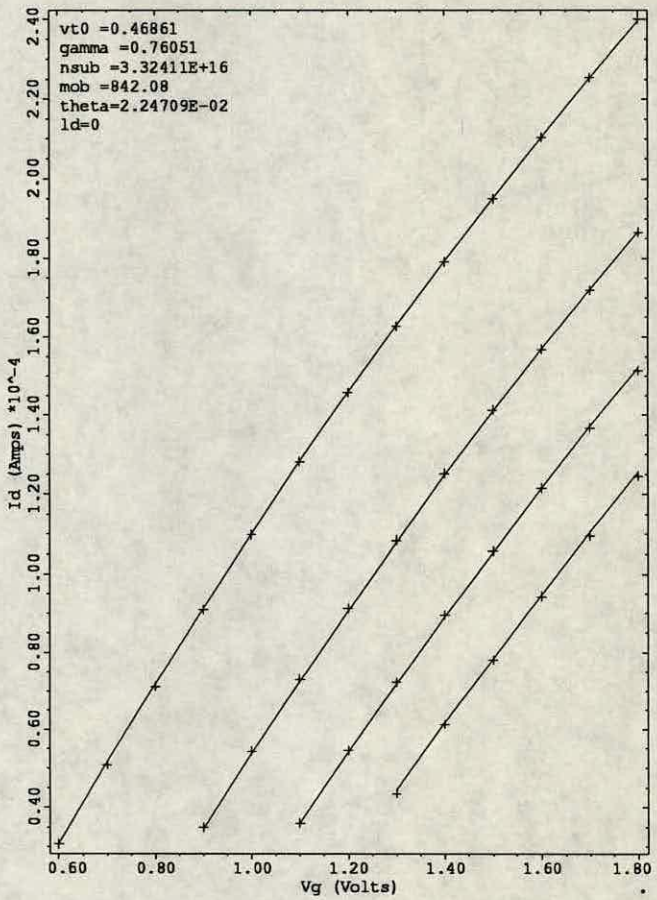


Figure 3–6: Topex fit showing unrealistic value for Ld

3.4 Integrated CAD Tools

With the arrival of cheaper computing hardware and user friendly environments, integrated CAD tools are becoming more and more important in the design engineer's tool kit. Essentially these tools provide an integrated environment from process simulation through to circuit simulation. The process flow is outlined in figure 3–8.

As Dutton describes in [40], there are four main requirements for an integrated CAD system to ensure acceptability in the semiconductor industry.

- Accuracy

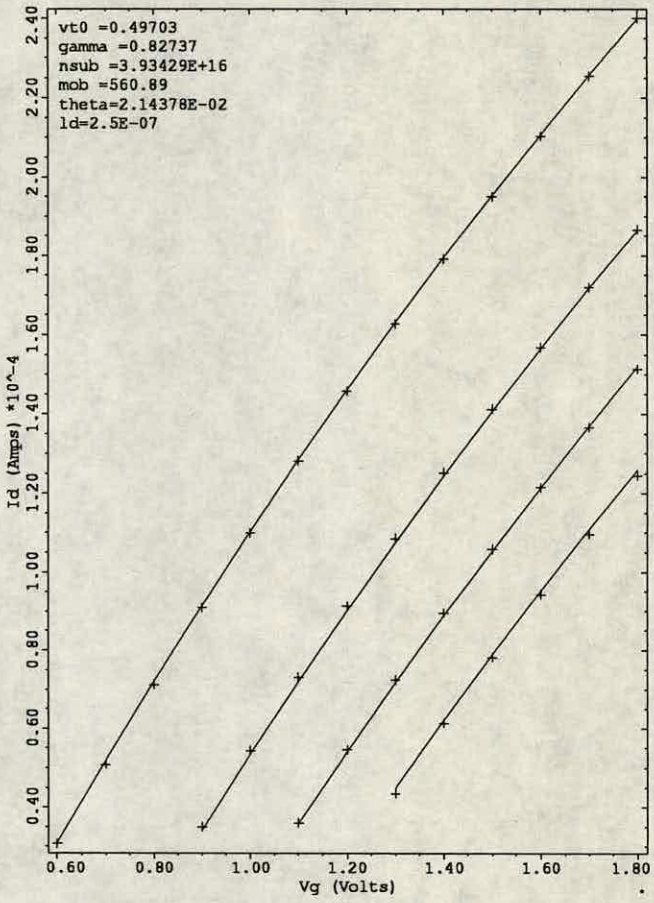


Figure 3–7: Topex fit showing realistic value for Ld

- Speed
- Friendliness
- System Integrability

Accuracy Since there is a theoretical loop from process through to circuit simulation, it is essential that the output passed from one program to the input of another is as close to reality as possible. This is established by calibrating all the simulators, whenever possible, and using the most suitable models available.

Speed As is the case in all companies, time means money and thus the faster the operation of such an integrated system, the faster the results can be filtered

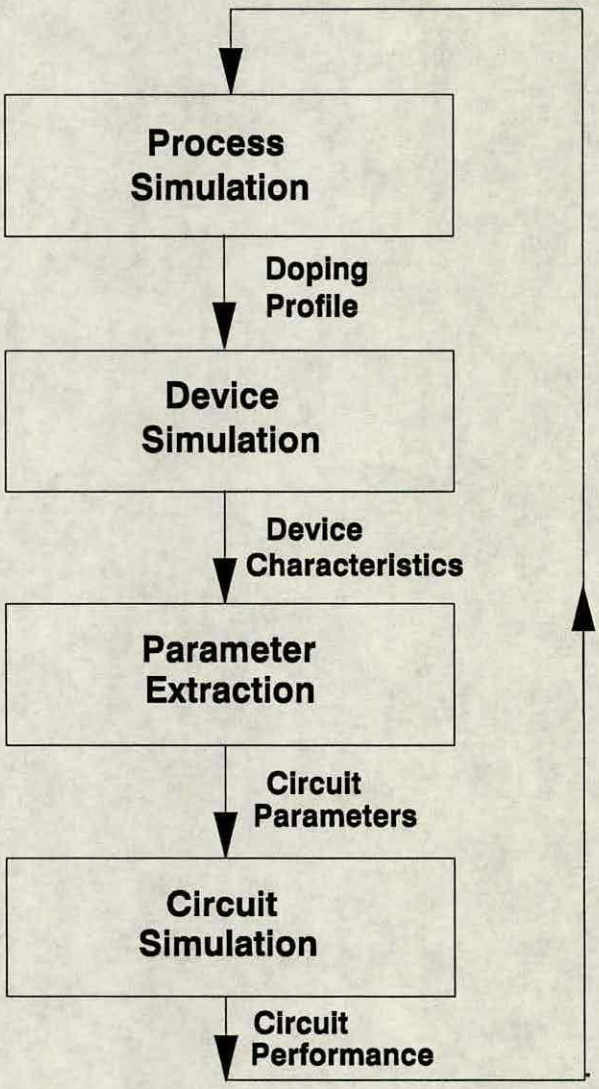


Figure 3–8: Integration of process, device, parameter extraction and circuit simulation tools

through to the fabrication line. Also important is the speed up in device development. If a device has a market window of 2 years and it can be brought on line 6 months earlier, then this can result in a significant increase in profit margins.

User interface and system integrability In order to facilitate ease of use and also to contribute to accuracy, the user interface between programs is important, especially between process and device simulation. System integrability is also a crucial factor for flexible use of such a tool in the future.

There are many and varied CAD tools available at present. OYSTER, [41], is a 3D visualization tool for device topology, SIMPL, [42], looks at process and device integration and PRIDE, [43], is a windows based integrated system within a device framework. Other tools include EASE, [44], and PEW, [45].

There is also available commercial CAD tools, two of which will be outlined in more detail. These tools are marketed by Silvaco International and Technology Modeling Associates (TMA).

Silvaco International's framework is shown in figure 3–9. Masterpiece provides the environment to perform all the necessary simulations for characterisation of both the fabrication process and device analysis. The results of this can be linked by dbVISTA, a commercially available database, to Vyper. This is used to perform extraction of circuit parameters and for performing circuit simulation and statistical analysis. The result is a highly automated design environment for CAD analysis.

The package available from TMA is called *studio*. This provides the user with an integrated CAD framework for semiconductor analysis, running under a windowed environment. The basic structure of *studio* is outlined in figure 3–10. The flexible system allows addition of new programs, modules as and when required.

As a result of such tools the process device and circuit engineer can not only shorten the development time for new processes, thus saving money, but also understand a great deal more about the device or circuit in question. Without such tools the introduction of new sub-micron processes will never be possible in the present timescale allotted. Thus for semiconductor companies to be competitive in the modern market, an integrated CAD environment is no longer a luxury but a necessity.

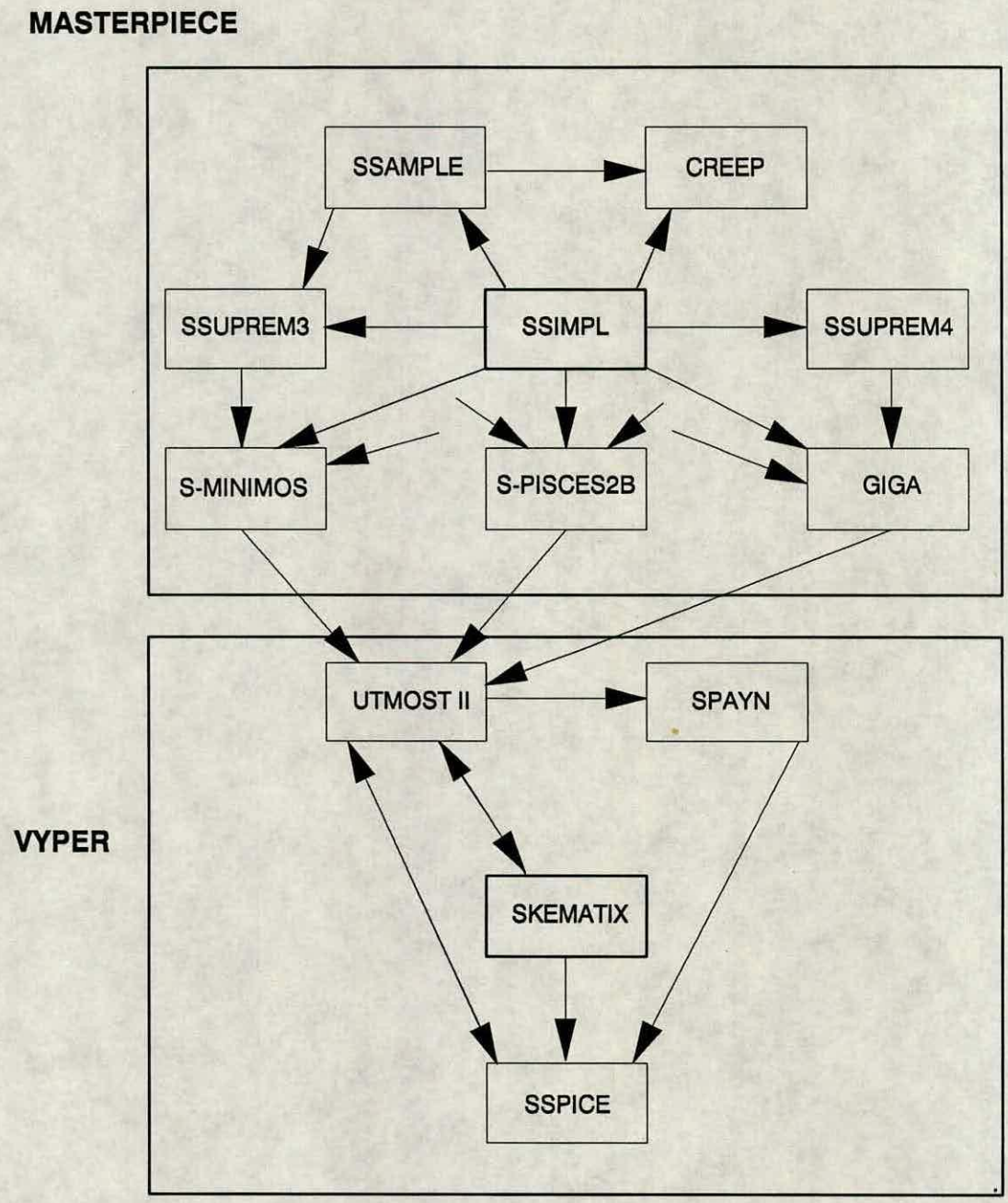


Figure 3–9: The CAD environment offered by Silvaco International

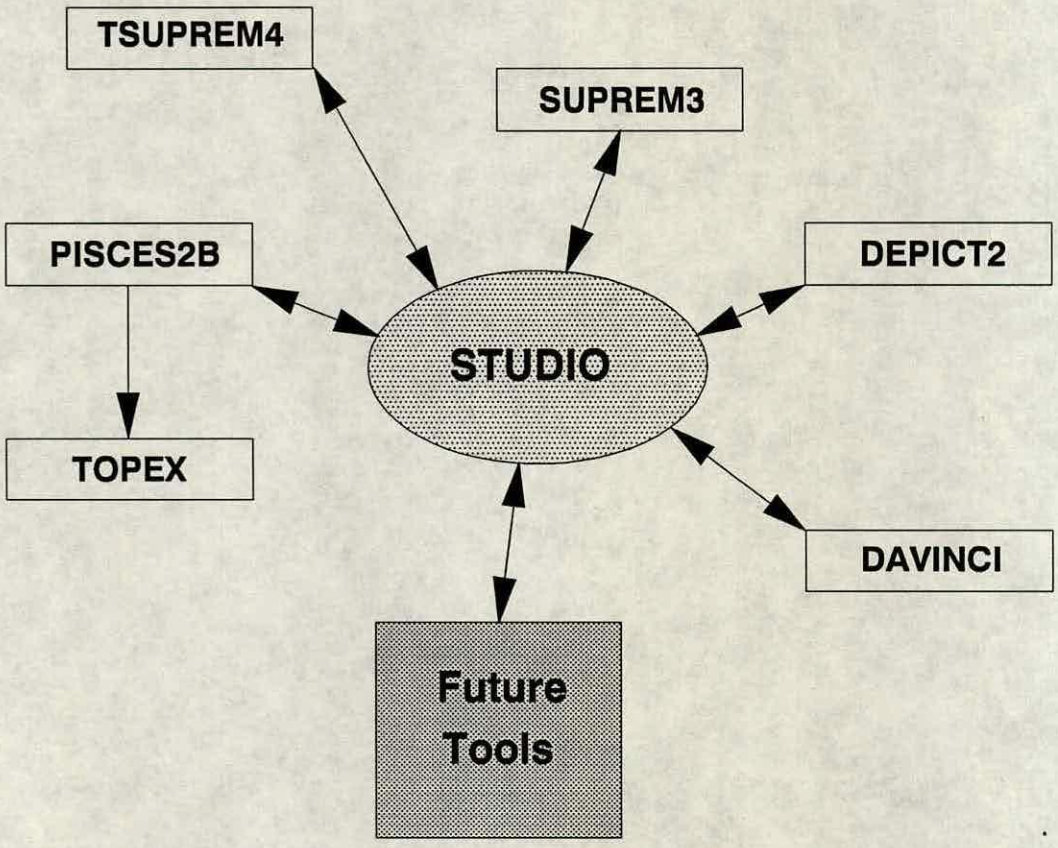


Figure 3–10: The CAD environment offered by Technology Modeling Associates

Chapter 4

Experimental Design and Process Optimisation

4.1 Introduction

Experimental design, or design of experiments, is a systematic approach of experimentation, designed to gain as much information as possible from a reduced number of experiments. These techniques have been outlined in detail in [46] and [47]. Because the approach is systematic, it allows engineers and scientists to collate data in such a manner that maximises the interpretation of these results. It is, however, not enough to be only systematic, as a "one variable at a time" approach does not always guarantee meaningful interpretation.

It is usual to collect data about a particular process by varying parameters over a specified range of operating conditions. The information from this data is then used to form a model of the process. The more accurate the data and interpretation techniques, the more reliable the model is likely to be. Statisticians are often guilty of deciding what to do with the data, rather than also considering how the data is collected, as Hicks [47] states, "*Often the experimenter is so anxious to collect data, stuff it into a neat little formula $\cdots$ and draw some conclusions that he pays no attention to how the data were collected*".

The experimental design process can be concisely summarised into the following steps.

- Determine the input variables or control factors which are to be suspected to influence the output variables or responses of interest. A traditional example is in the realm of agriculture, where the control factor could be fertilizer and the output response crop growth. The number of input variables at this stage could be numerous.

  In the area of semiconductors possible input factors could be implant dose and energy or oxidation time and temperature. The responses could include threshold voltage and oxide thickness.

- Based on engineering judgement determine the range of operating conditions for the control factors and the responses. This is often referred to as the parameter space.

- Perform screening analysis. Experiments should be carried out that will determine the first order effects of all the control factors. This reduces the potential number of factors to a more manageable size. The results from screening can form the basis of more detailed second order designs.

- Choose an appropriate statistical design, (see section 4.2) and carry out experiments as decided by the design in question.

- Fit a model to the data.

- Analyse the results using statistical tests and analysis. Results should also be considered from an engineering standpoint.

Figure 4–1 illustrates the essential objective of experimental design techniques: The determination of what will happen to the output responses (dials) for changes in the input factors (knobs) for a given process. The process can be considered as the transfer function of control factors to responses. Often, based on previous engineering knowledge, it may be possible to determine how a particular response varies for a given single input factor. It is, however, at best difficult and often impossible to determine how "the process" will dictate the response values for

simultaneous variation of a number of input factors at a time. The advantage of fitting a model to the process, means that once the original experiments have been performed, the knobs (inputs) can be changed to any value within the parameter space and the values of the dials (responses) can be immediately established.

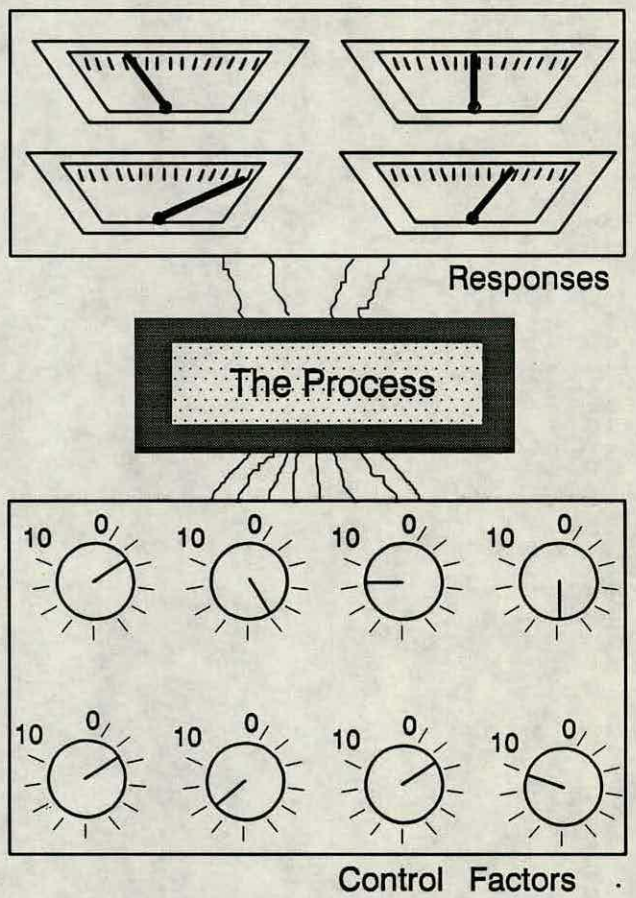


Figure 4–1: Graphical representation of the experimental design process

In general experimental design can be used for a variety of broad applications. It can be used in the initial design stages of a new process or product, eg designing a new semiconductor fabrication process. It can also be used to determine which factors affect a process or responses most significantly, eg screening analysis. A popular application is in the optimisation of an existing process, eg reducing the sensitivity of a response to particular input factors. This optimisation application will be outlined in more detail in chapter 7.

4.2 Statistical Methods

There are available at present a number of statistical methods that can be used in the design of experiments. They are what may be determined experimental objectives and as the name suggests, the choice will depend on the purpose or objective of the experiment. Three of the most common objectives are Screening, Response Surface Methodology (RSM) and Taguchi. These will be outlined in more detail in the following sections.

4.2.1 Screening

As previously stated, screening experiments are essentially used to reduce a large number of potentially important factors to a more manageable size. Not all factors will exert the same level of influence on the output responses, it is therefore possible to identify a factor subset that will have a much greater influence on the responses than the others.

Initial experiments are usually limited to relatively simple two-level designs in order that the most significant factors can be efficiently determined. It is assumed that for screening purposes interactions are ignored. Two factors are said to interact when the effect of one factor on the response depends on the level of the other factor. Figure 4–2 shows examples of interaction effects.

Two of the most common statistical designs used in screening experiments are fractional factorial and Plackett-Burman [46]. A full factorial experiment, ie all permutations and combinations of the input factors, is usually too costly for screening purposes, hence the need for a fractional factorial experiment. Figure 4–3 shows a fractional factorial design for three variables and two levels.

A popular fractional factorial design is a half factorial or $2^{(n-1)}$, which is a specific type of fractional design governed by the relationship $2^{(k-p)}$. For example, for screening five variables at two levels a factorial design would require 2^5 or 32 experiments, whereas a half factorial would only require 2^{5-1} or 16 experiments.

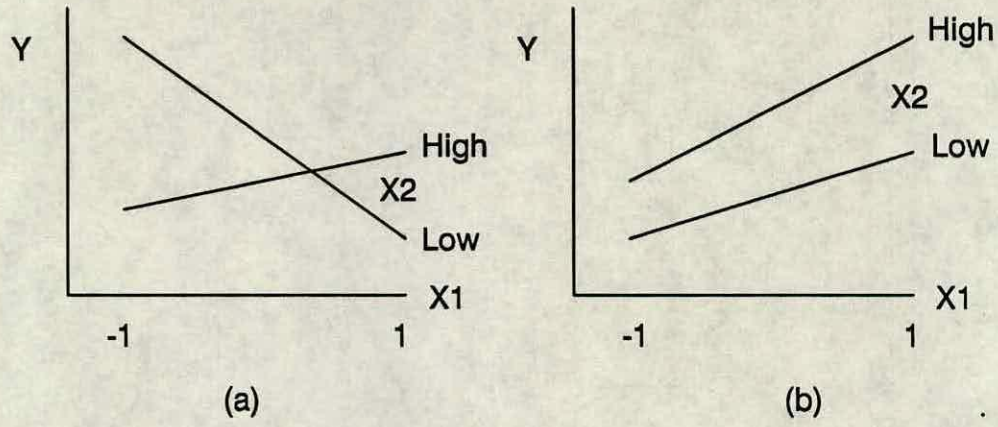
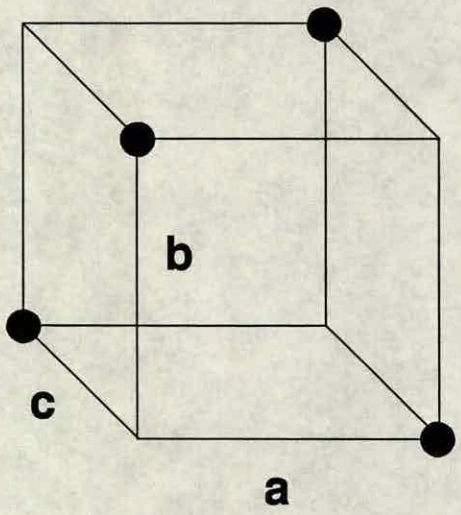


Figure 4–2: The comparative effects of interactions and no interactions



a	b	c
+1	+1	+1
+1	+1	+1
-1	+1	-1
-1	-1	+1

Figure 4–3: A fractional factorial design for 3 variables and two levels

The second design used commonly in screening experiments is the Plackett-Burman design. This requires a very small number of experiments, where the number of runs is in multiples of four ie 4, 8, 12, etc. For example, an experiment with 11 factors can be carried out with only 12 runs. The Plackett-Burman design works on the principle of the two-factor interactions being averaged out over a large number of main effects and not just a single one. Assuming that two-factor interactions are small compared to main effects, then any offset in a main effect due to this interaction will also be small.

4.2.2 Response Surface Methodology

Once the significant factors and proper level settings have been established, it is necessary to use Response Surface Methodology (RSM) techniques to arrive at an optimum level for the response or responses [48]. RSM was defined in Myer's review, [49], as "*a collection of tools in design or data analysis that enhance the exploration of a region of design variables in one or more responses*".

Essentially RSM seeks to answer the following questions [46].

- How does a specified response vary with change in the input variables, over the given region of interest?
- What settings, if any, will result in simultaneously meeting all the specified requirements?
- What values of input variables result in a maximum or minimum response and what does the surface look like at this point?

The method which RSM uses to answer these questions is essentially to perform experimentation at the specified set of conditions, fit the responses to a mathematical function and statistically evaluate the quality of the model with the experimental data. It is the plotting of contours for the response function that constitutes the response surface. An example of a response surface or contour plot is illustrated in figure 4–4

There are different forms of response surfaces which are considered as a typical representation of the type of plots that are obtained in practice. These surfaces are outlined in figure 4–5.

The plots of figure 4–5 can also be used to illustrate the problems associated with the traditional one-factor-at-a-time approach at reaching the optimal point of the response surface. In figure 4–5a, if X_2 is fixed and X_1 is varied, it is possible to find the optimum value of the response, Y, with respect to X_1. By fixing this value of X_1 and varying X_2 one can arrive at the optimum value of Y with respect

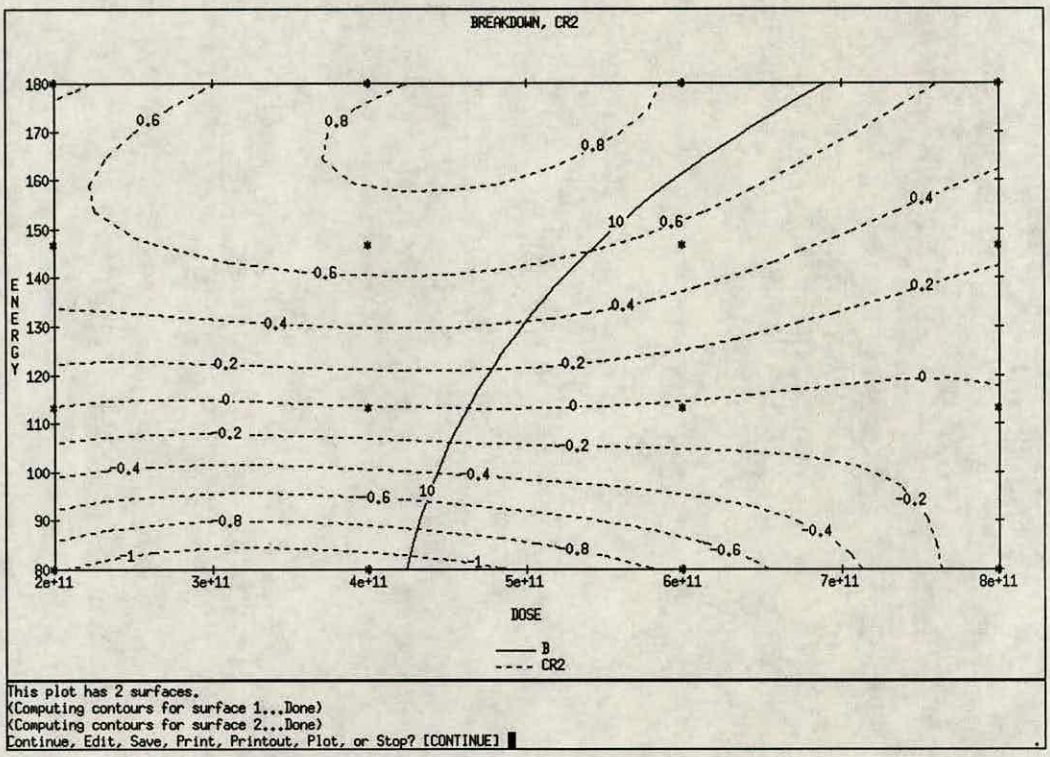


Figure 4–4: An example of a typical response surface

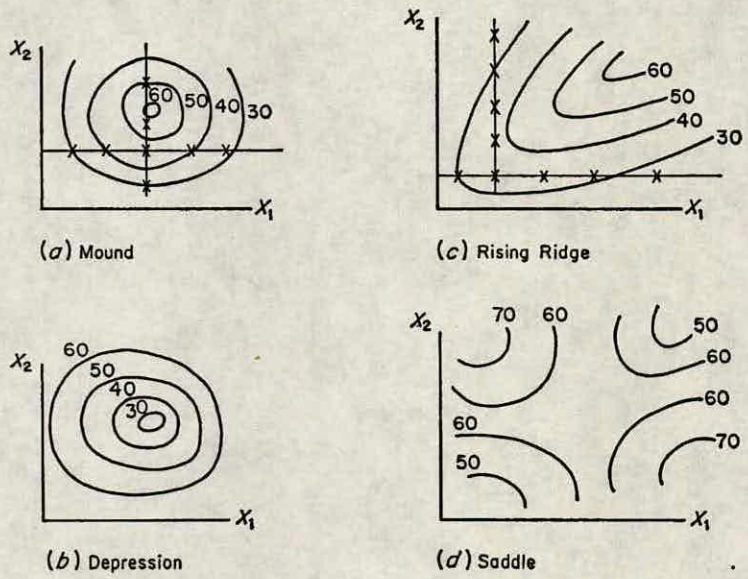


Figure 4–5: Typical response surfaces in two dimensions

to both X_1 and X_2. It is not possible, however, to arrive at the optimum value for figure 4–5c by using this approach, hence the need for RSM techniques.

The fitting of the mathematical function to the desired response is crucial to the validity of the final results obtained. It is assumed that a response, Y, is a function of a set of input variables or factors $x_1, x_2, ..., x_k$ and that the function can be approximated in some region by a polynomial model. The general form for a first-order model is:

$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon \tag{4.1}$$

and for the second-order model:

$$Y = \beta_o + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x_i^2 + \sum_{i=1}^{k} \sum_{j=1}^{k} \beta_{ij} x_i x_j + \epsilon \tag{4.2}$$

Considering the example of two variables with two levels of input the first-order model becomes:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon \tag{4.3}$$

and the second-order model becomes:

$$Y = \beta_o + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \beta_4 {x_1}^2 + \beta_5 {x_2}^2 + \epsilon \tag{4.4}$$

Here the $x_1 x_2$ term represents the interactions between x_1 and x_2 and ϵ is the error term. For simulation, the error term is due to model error, whereas for experimental analysis the error term is due to both random and model errors. The coefficients β_0, β_1 etc can be calculated using the method of least squares [47].

In a similar manner the cubic model can have the following form where x_1 and x_2 are two different input factors.

$$\begin{aligned} Y \;=\; & \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_1^2 + \alpha_4 x_1 x_2 + \alpha_5 x_2^2 + \alpha_6 x_1^3 + \alpha_7 x_1^2 x_2 + \\ & \alpha_8 x_1 x_2^2 + \alpha_9 x_2^3 + \epsilon \end{aligned} \tag{4.5}$$

This model will be used in the analysis outlined in chapter 7. Each of the terms in x_1 and x_2 have been transformed to ensure that the factor settings are standardised to a common range to facilitate the relative evaluation of the terms in the model. The transformation is of the following format and is known as orthogonal coding.

$$\hat{x} = \frac{x - mid}{\frac{1}{2}r} \tag{4.6}$$

Where $\hat{x}$ is the transformed value of x, mid is the midpoint of the range of x and $\frac{1}{2}r$ is half the range of x.

Just as Plackett-Burman and fractional factorial designs are normally used for screening experiments, so RSM has traditional designs associated with it. These designs include Central Composite, D-optimal and Box-Behnken designs [49].

It is the fundamental properties of these designs to reduce the number of experiments required compared with full-factorial analysis. Each of these designs will be outlined in more detail.

4.2.3 Central Composite Designs

These designs are essentially extensions of two-level full factorial designs, with the addition of axial (or star points) and centre points. They can be best illustrated by considering figure 4–6.

This shows how the central composite design is made up of its constituent parts, namely the cube portion, the axial portion and the centre points. Parameter a can vary depending on the type of design.

There are three basic types of central composite designs; Central Composite Circumscribed (CCC), Central Composite Inscribed (CCI) and Central Composite Face-centred (CCF).

Central Composite Circumscribed This is termed a spherical design, where the cube portion is at the factorial points, the axial portion has values of factors outside the factor parameter space. Thus this design cannot be used in situations where the factor ranges represent strict limits. This design is outlined in 2 dimensions (for clarity) in figure 4–7.

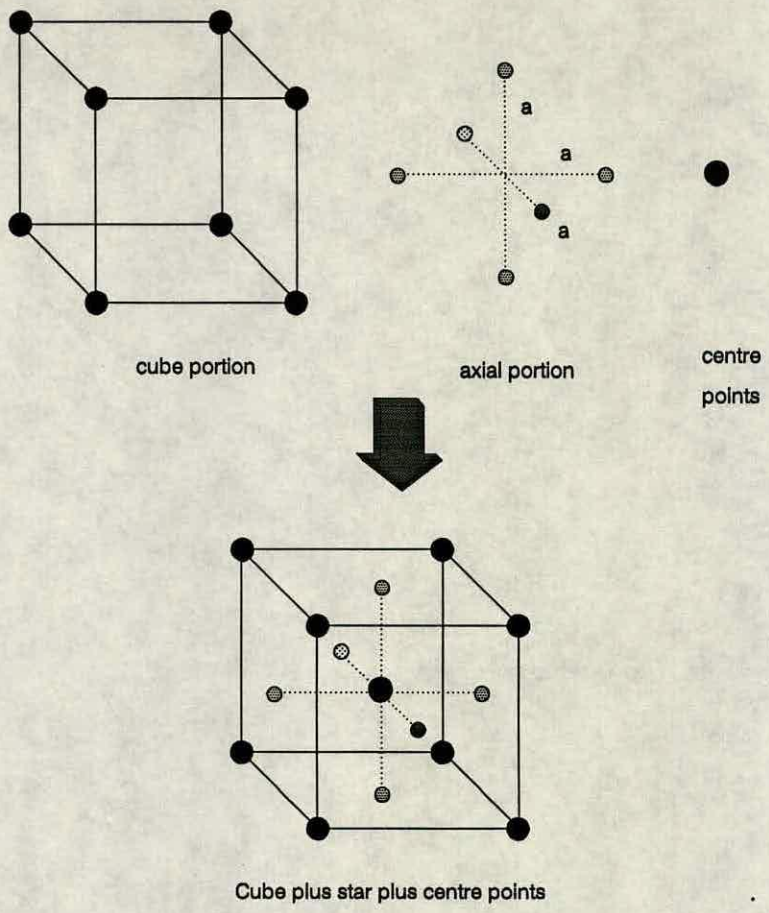


Figure 4–6: A three-dimensional central composite design, showing cube portion, axial portion and centre points

Central Composite Inscribed This is also a spherical design, similar to CCC. In this case, however, the axial points are at the limits of the factor ranges. This results in the factorial points being no longer at the endpoints of the cube. See figure 4–7.

Central Composite Face-centred The cubic portion for this design is at the factorial points (as for CCC). The axial points, however, are placed along the face of the cube. This is termed a cubic design. Thus all points are within the factor parameter space. See figure 4–7 for a comparison (in 2 dimensions) with CCC and CCI.

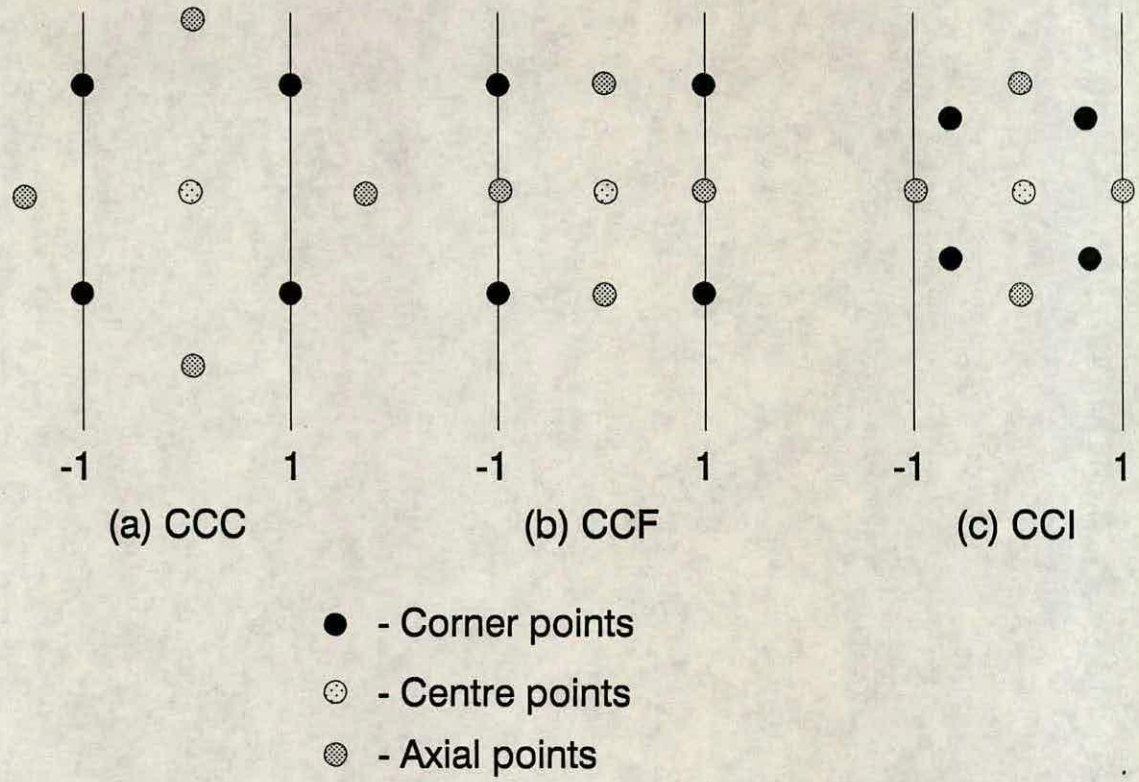


Figure 4–7: A two dimensional comparison of CCC, CCI and CCF Central composite designs

4.2.4 D-optimal Designs

Although not considered one of the "classical" designs, this design is both powerful and flexible in the field of RSM. The detailed theory of D-optimal designs is outside the scope of this work but can be found in [48], [50]. A brief overview, however, will be given below.

If we consider a linear model.

$$Y = X\beta + \epsilon \tag{4.7}$$

Where Y is a matrix of responses, X is a matrix of the modelling terms, β is a vector of model coefficients and ϵ is a matrix of residuals with variance σ^2. The D-optimal criterion, from theory, is defined as maximising the determinant of the matrix $X'X$, where X' represents the transpose of the matrix X. This

results in the minimisation of the variances of the estimated model coefficients. The determinant can be increased either by increasing the number of runs or else by increasing the range of the factors. Thus, for a fixed range and number of runs, the problem is to find the largest determinant, the D-optimal design.

According to [51], D-optimal designs have the following advantages:

- They can minimise the variance of the model coefficient for a given number of runs.

- The number of experimental runs can be chosen depending on outside constraints.

- The design may be used in stages, with one design leading to another.

Another important feature of D-optimal designs is the ability to specify exclusions. This simply refers to the procedure of designating certain areas of the design that are not practically possible, ie values which are out of specification, or negative values for terms that can only be positive. An example of this in the semiconductor area would be a negative junction depth for a particular set of input conditions.

An example of a D-optimal design for a 1.5μm MOS process will be outlined in section 7.4.1.

4.2.5 Box-Behnken Designs

These designs are three level in nature and are essentially spherical. The current operating conditions can be considered as the centre of a cube, and the values for the other factors are placed midway between all the cube vertices. There are, therefore, no points at the actual cube vertices.

4.2.6 Taguchi Methods

These methods originated from the Japanese engineer Dr Genichi Taguchi. They are concerned with reducing the variability of a product, while at the same time reducing costs and increasing yield. Quality according to Taguchi, should not be achieved through inspection, but should be pushed back to the original design stages. Thus the final product should be operating at its optimum level, with a minimum sensitivity to changes in input factors.

The Taguchi method is considered strong as it introduces a structural approach to experimentation and since it considers sensitivity of the output it has the dual effect of centering the design and minimising the response sensitivity to input fluctuations or noise. This is outlined in more detail in [52], [50]. It has also been criticised though as inflexible and not always dealing adequately with interactive terms [49, 8]. It is also not always very efficient in the number of experimental runs that are required.

The Taguchi structure is made up of inner arrays and outer arrays. The inner array corresponds to the level settings of the control factors, whereas the outer array corresponds to the settings of the noise factors for each setting of control factors. Although much information can be obtained from this type of analysis, it has a drawback of requiring large amounts of experimental runs. For example, if we consider an inner array of four factors and three levels and an outer array of three factors and two levels, this would result in $3^4 \times 2^3$ ie 648 experimental runs.

To compensate for this, however, Taguchi introduced the concept of Orthogonal Arrays. These arrays were coded by Taguchi and made available to the engineer for experimentation [53]. These arrays sacrifice secondary information that would normally be available from a full factorial design. The nomenclature normally used for these arrays is $L_x z^y$ where x is the number of experimental runs, z is the number of level settings and y is the number of input factors. Figure 4–8 shows how inner and outer arrays can be used with orthogonal arrays.

As can be seen the orthogonal design results in only 36 experimental runs compared with the 648 described previously.

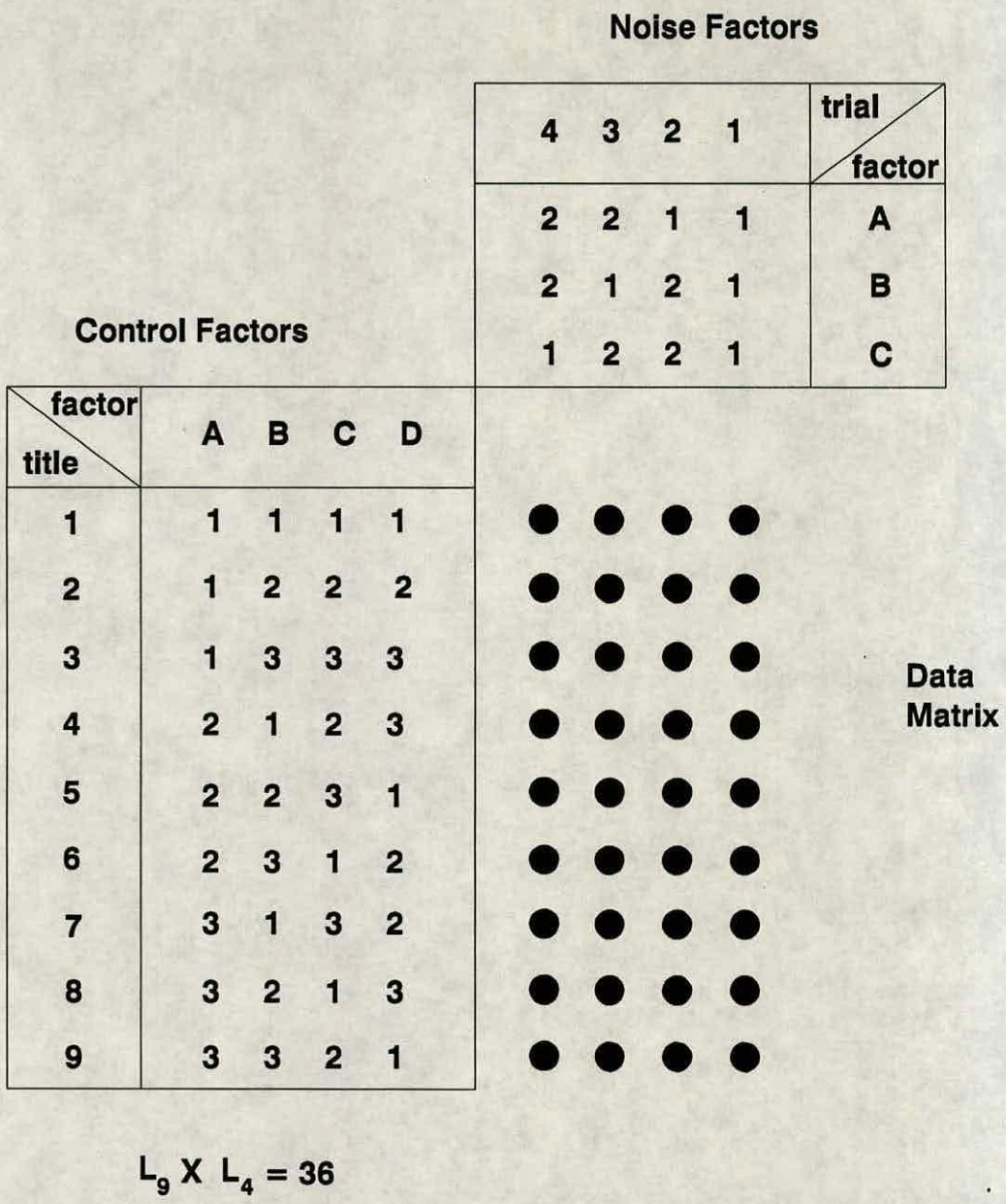


Figure 4–8: Taguchi inner and outer arrays for a $L_9 \times L_4$ orthogonal design

The results from these experiments are then analysed on two accounts, namely the signal-to-noise ratio and mean for each output response. This gives a measure of both the variability of the response and its actual or target value.

Depending on the type of analysis to be performed, Taguchi proposed three main types of signal-to-noise ratios [52]. For all three ratios the output is least sensitive to noise variations when the signal-to-noise ratio is at its maximum value. These three ratios are outlined below.

- Target is best.
- Larger is better
- Smaller is better

Target is best As the name suggests this is used for situations where a specific target value is required. The signal-to-noise ratio is defined as follows:

$$SNR = 10\log_{10}\frac{\mu^2}{\sigma^2} \tag{4.8}$$

Where μ is the mean value of all the observations and σ^2 is defined as

$$\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \mu)^2 \tag{4.9}$$

where n is the number of observations and y_i the i th observation.

Larger is better This is used for situations where the required response is to be maximised. The signal-to-noise ratio is defined as:

$$SNR = -10\log_{10}\frac{1}{n}\sum_{i=1}^{n}\frac{1}{y_i^2} \tag{4.10}$$

Smaller is better This is used for situations where the required response is to be minimised. The signal-to-noise ratio is defined as:

$$SNR = -10\log_{10}\frac{1}{n}\sum_{i=1}^{n}y_i^2 \tag{4.11}$$

Figures 4–9 and 4–10 show the plots obtained for signal-to-noise ratio analysis and mean analysis respectively, for a semiconductor example. The four control factors were threshold adjust implant dose and energy and oxidation time and temperature. The output responses were threshold voltage and oxide thickness. The figures show the results obtained for threshold voltage, similar plots were also

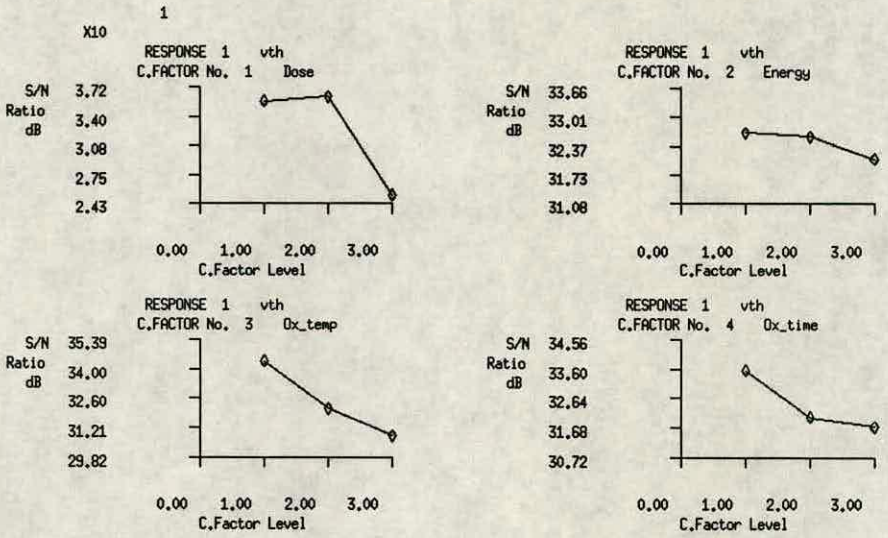


Figure 4–9: Signal-to-noise ratio graphs for threshold voltage

obtained for oxide thickness. From figure 4–10 one can arrive at an appropriate set of conditions to result in an acceptable level of threshold voltage. Figure 4–9 can then be used to establish if these conditions result in an insensitive, robust design. For example if a high value of threshold voltage (Vth) is required it would be more beneficial to use a low value of energy rather than a high value of dose to achieve this, since a high dose results in a low signal-to-noise ratio. The signal-to-noise ratio used in this situation was target is best.

A comparison of RSM and Taguchi methods will be outlined for an example of a 1.5 μm nMOS process in chapter 7.

4.3 Experimental design and semiconductors

Experimental design has traditionally been used in chemical and processing applications [54, 55], but it has now become popular in the field of semiconductors [56-59, 6, 60, 7]. With these techniques it has been possible to obtain optimum levels for the settings of various processing parameters, eg furnace temperature, implant dose and energy. Using sensitivity analysis the levels have also been chosen

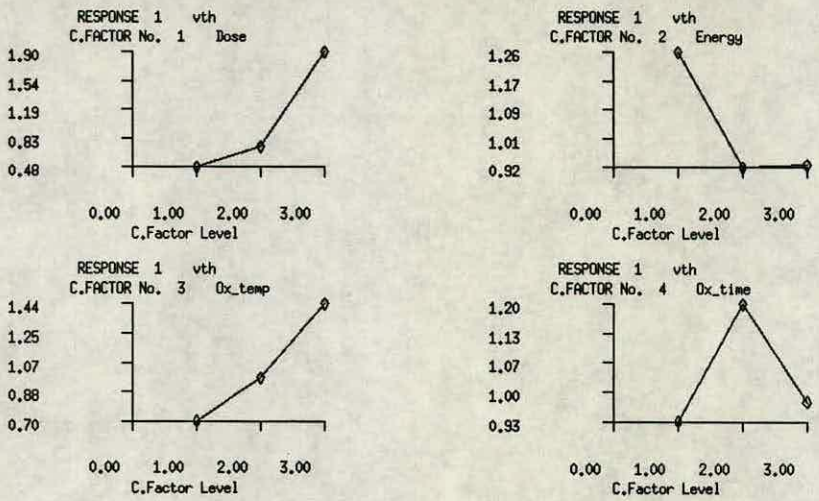


Figure 4–10: Mean response graphs for threshold voltage

to make the output responses as insensitive as possible to fluctuations or noise in the input factors. Experimental design techniques can also be used very effectively with the simulation software described in chapters 2 and 3. Instead of carrying out experiments, simulations can be performed. This leads to faster results at a reduced cost, thus improving the efficiency of the technique.

One of the "in" phrases in industry at present is Design for Manufacturability or DFM. William Miller of Harris Semiconductor described DFM as "a methodology which supports the design of products which are inherently producible within a target manufacturing environment." So often in the semiconductor industry engineering decisions are based largely on the engineering judgement of the engineer in question. DFM, however, replaces intuition with a formal design method. Essentially this design method means carrying out some or a combination of all of the design methods outlined in section 4.2. The ultimate goal is to increase the yield of the product, thus decreasing costs, hence increasing profitability.

4.3.1 Tools of the trade

The software environment where the experimental design or DFM was carried out is called RS/1 [61]. This program will be outlined in more detail in the subsequent sections, along with some simple examples from a typical semiconductor fabrication situation.

RS Discover

This is the program within RS/1 that allows the engineer to set up the experiments according to the user specifications.

The procedure for designing an experiment within RS/1 is outlined below [62]. The simple example described earlier for Taguchi analysis, will now be used to illustrate the features of RS/1.

1. Define objective
2. Define input or control factors
3. Define the output responses
4. Specify a model
5. Specify a statistical design
6. Run the experiment
7. Analyse and interpret the data

These will be outlined in more detail below:

1. Define objective This involves making the decision regarding the type of experiment to be performed. The choice of objective is from; screening, response surface methodology and Taguchi designs (see section 4.2).

2. Define input or control factors The next step in the process is the choice of factors which have influence on the output responses. For the example of the MOS transistor, the input factors were channel implant dose and energy and oxidation time and temperature.

3. Define the output responses Any number of responses can be defined, along with their required range. For the given example the responses of interest were threshold voltage and oxide thickness.

4. Specify a model This is the generic form of equation that will be used to model the response surface. The three basic types of models are `linear`, `quadratic` and `cubic`. The higher the order of the model the more levels will be required for the input control factors. The default (and chosen model in this case) was quadratic.

5. Specify a statistical design This is a function of the previous steps described above. These have been detailed already in section 4.2. The CCF design was taken as the required design for the analysis of the MOS transistor.

6. Creating a worksheet From the information in steps 1-5 a worksheet can be created automatically by RS/1. This contains a list of all the experimental runs that need to be performed. An example of a worksheet is shown below.

Experiment Worksheet

1 ENERGY (KeV)	2 DOSE (atoms/cm2)	3 OX_TIME (mins)	4 OX_TEMP (deg C)	5 V_THRESH (volts)	6 OX_THICK (UM)
55.0	5e+12	60.0	1000	2.25	0.053300
50.0	2.8e+12	100.0	950	1.42	0.047002
50.0	2.8e+12	80.0	950	1.26	0.039726
45.0	5e+11	100.0	900	0.31	0.026524

55.0	2.8e+12	80.0	950	1.24	0.039727
50.0	2.8e+12	80.0	950	1.26	0.039726
45.0	2.8e+12	80.0	950	1.31	0.039726
55.0	5e+11	60.0	900	0.20	0.018175
55.0	5e+12	100.0	900	1.23	0.026520
50.0	2.8e+12	80.0	950	1.26	0.039726
50.0	2.8e+12	80.0	950	1.26	0.039726
50.0	5e+12	80.0	950	1.82	0.039722
45.0	5e+12	100.0	1000	2.90	0.078242
45.0	5e+12	60.0	1000	2.28	0.053299
55.0	5e+11	100.0	900	0.29	0.026525

7. Run the experiment It is now necessary to perform the different experimental runs outlined in the worksheet. In the case of semiconductors these runs can be real experiments using wafer fabrication (see chapter 8) or simulation of the experimental data (chapter 7).

8. Analyse and interpret the data Once the experiments have been carried out, the results are input into the worksheet for analysis using graphical and statistical analysis, the latter being detailed in section 4.3.1. For the case of response surface analysis (which is the major method used in this work), the response or contour plot is a most powerful and graphic technique of interpreting the results. Figures 4–11, 4–12 and 4–13 show contour plots for the worksheet outlined previously. These plots show how threshold voltage, Vth and oxide thickness, T_OX vary with change in channel implant dose and energy and gate oxidation time and temperature.

Considering figure 4–11 as an example it can be seen how the oxide thickness varies only with oxidation time and not with dose (as would be expected). Examining threshold voltage, however, shows that it varies both with oxide thickness and implant dose. As the dose increases, and/or the oxide thickness increases, so

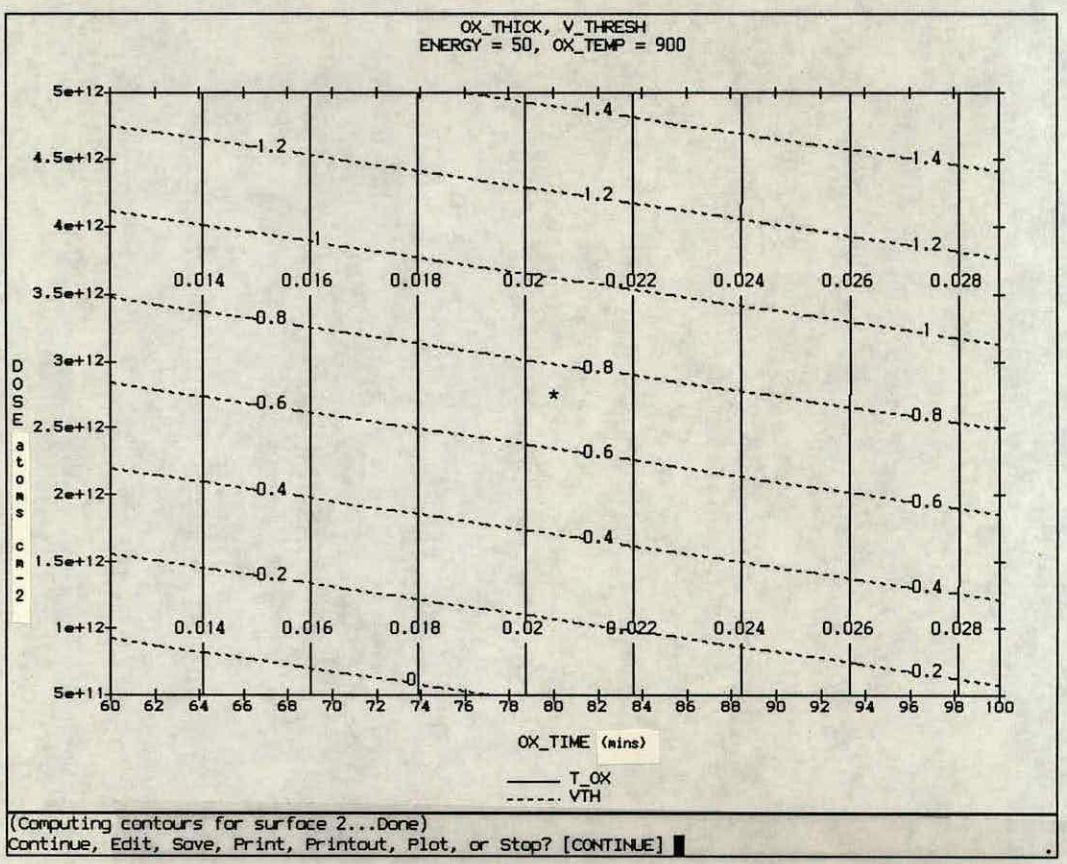


Figure 4–11: A contour plot showing the variation of Vth and T_OX with oxidation time and implant dose for the given values of energy = 50 keV and oxidation temperature = 900 deg C

does the threshold voltage. Another benefit of the contour plot is that values for oxide thickness and threshold voltage can be easily obtained for any set of control factor conditions within the parameter space. No further experimentation, or simulation is required.

The effects of oxidation temperature can be seen by comparison of the three figures. As the temperature increases, so does the oxide thickness and the threshold voltage.

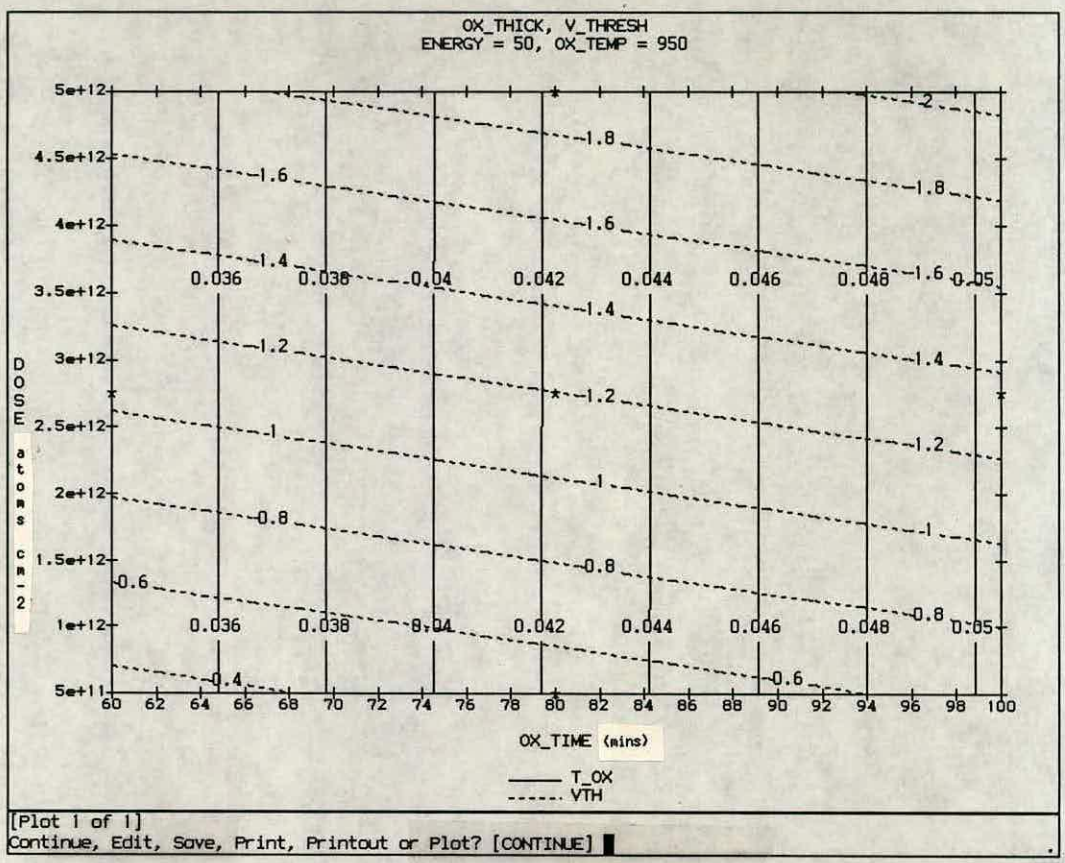


Figure 4–12: A contour plot showing the variation of Vth and T_OX with oxidation time and implant dose for the given values of energy = 50 keV and oxidation temperature = 950 deg C

RS Explore

This is the program within RS/1 that is responsible for statistical analysis of the results obtained. It allows the engineer to visualise and analyse data relationships, to provide better understanding of the process being modelled. The facilities available include; graphical plots eg scatterplots, histograms; variable transformations; analysis of variance (ANOVA) and tests for significance [63].

Some of these facilities which are necessary in the analysis of completed designs will be outlined in more detail. The major part of this analysis will be concerned with evaluating the fit of the model to the data. Within RS/1 there are essentially

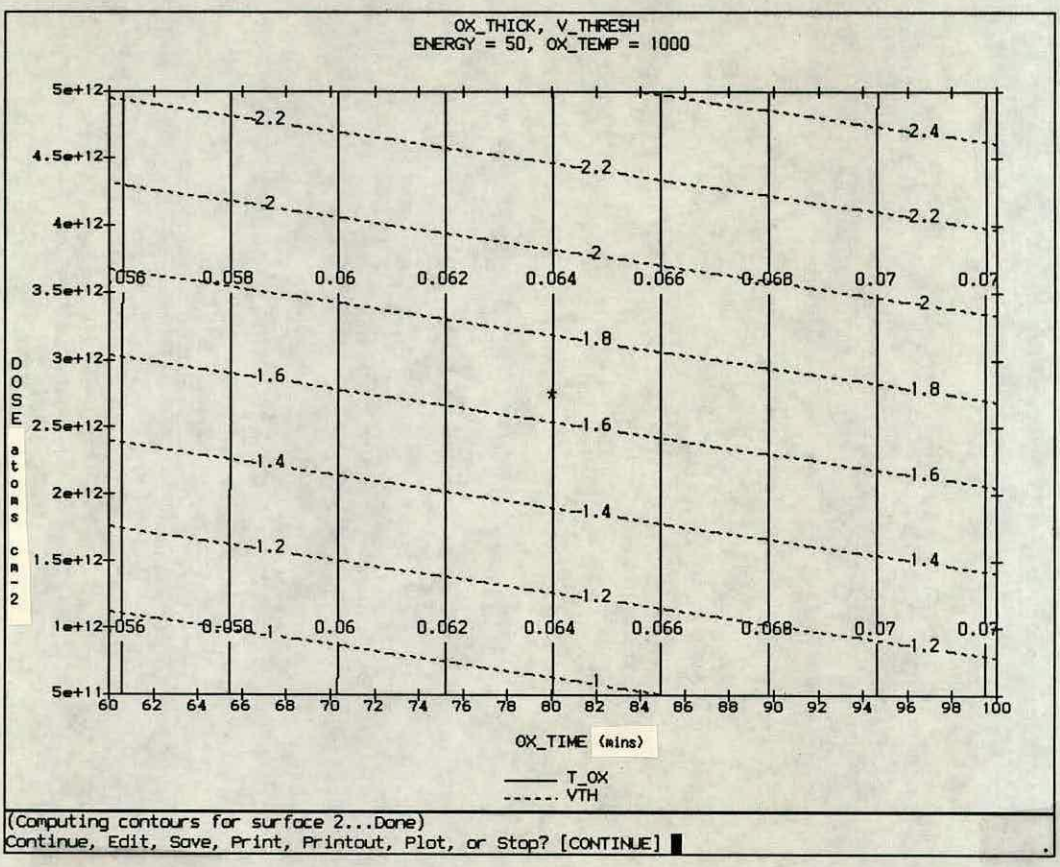


Figure 4–13: A contour plot showing the variation of Vth and T_OX with oxidation time and implant dose for the given values of energy = 50 keV and oxidation temperature = 1000 deg C

three methods of assessing the validity of the fitted model. They are presented in the form of a table and are as follows:

1. The residuals table.

2. The coefficients table.

3. The analysis of variance table or ANOVA table.

The residuals table This contains details of the factor values, the data value, the model value and the residual value at each setting. The residual is the difference between the data value and the model value. This information can be

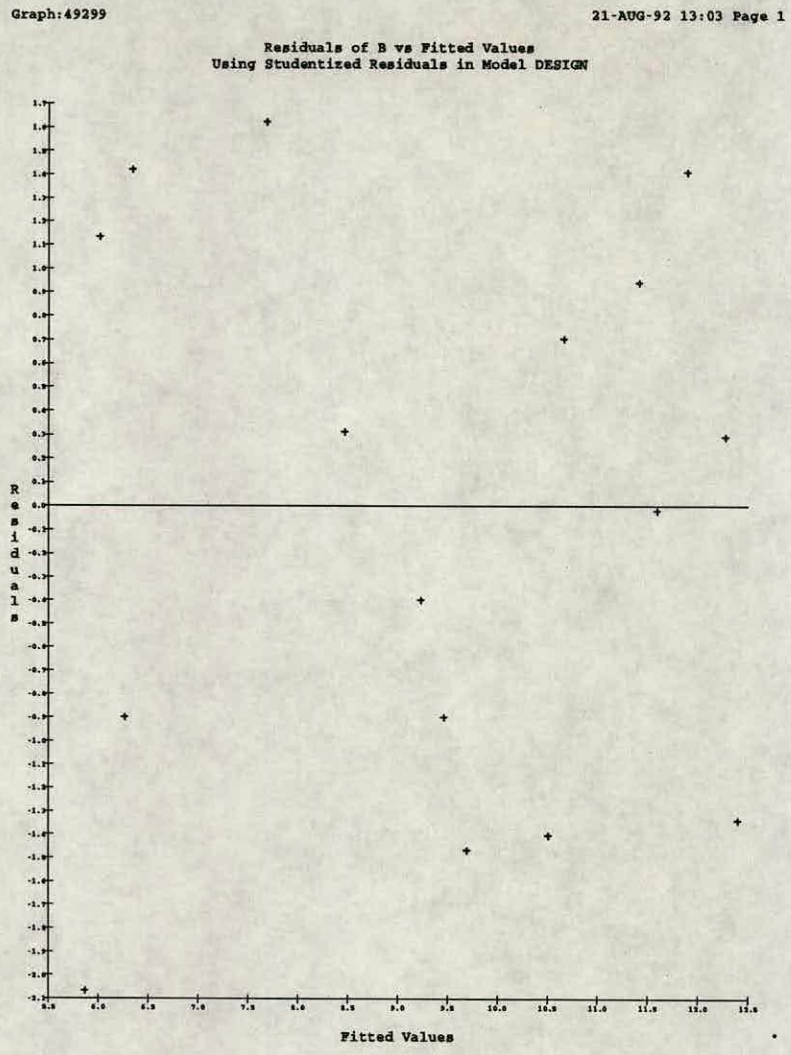


Figure 4–14: A randomly distributed scatterplot

presented in the form of a table, but it is often more concise to view the data in graphical format.

If the model adequately fits the data then the variation in the residuals should be random in nature. This can be illustrated by use of the scatterplot. The residuals should be randomly scattered above and below the zero line, thus indicating no trend or offset in the fit. A typical scatterplot is outlined in figure 4–14.

The normal probability plot of the residuals can also be used to analyse the distribution of the residuals. The points on the plot should roughly follow a

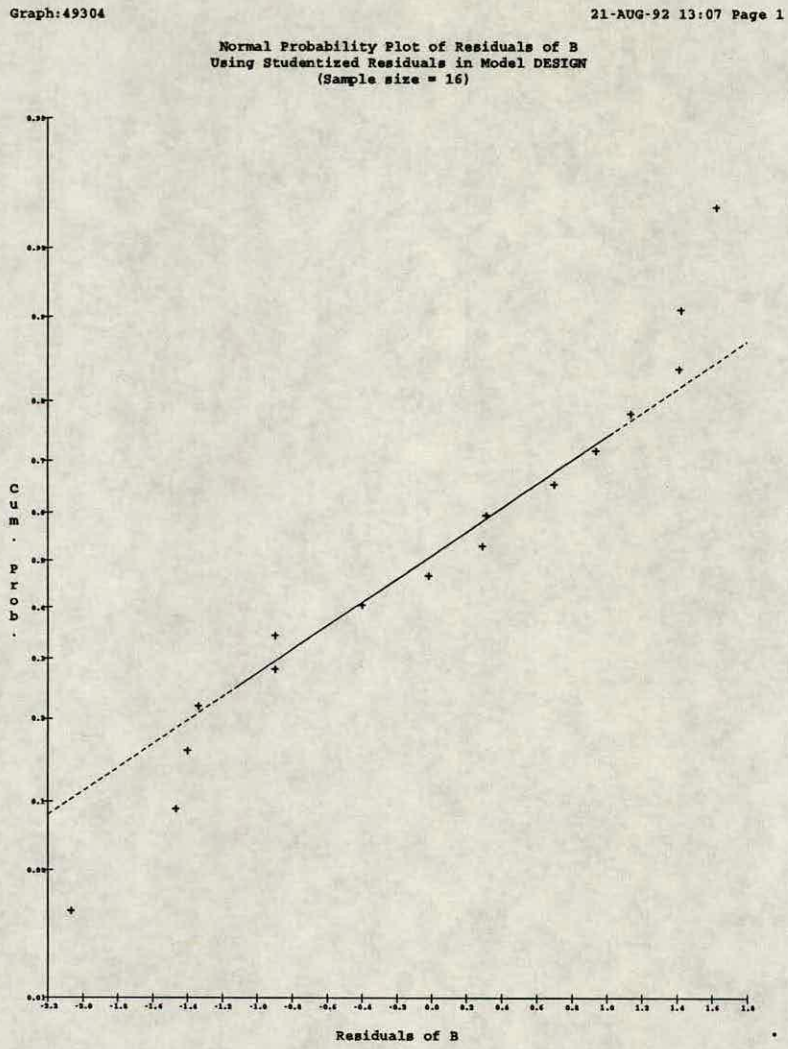


Figure 4–15: A normal probability plot. A straight line indicates a normal distribution.

straight line if the residuals are normally distributed. An example of a normal probability plot is detailed below in figure 4–15.

Coefficients table The coefficients table is used to decide which terms in the model can be considered significant with regards to their contribution to the model fit. The higher order coefficients have a *T-value* and a *significance* associated with them. The T-value of each coefficient is the test of the hypothesis that the coefficient is actually zero. The significance level is the probability of taking the coefficient as non-zero when in fact it is zero. Thus a high T-value means a low

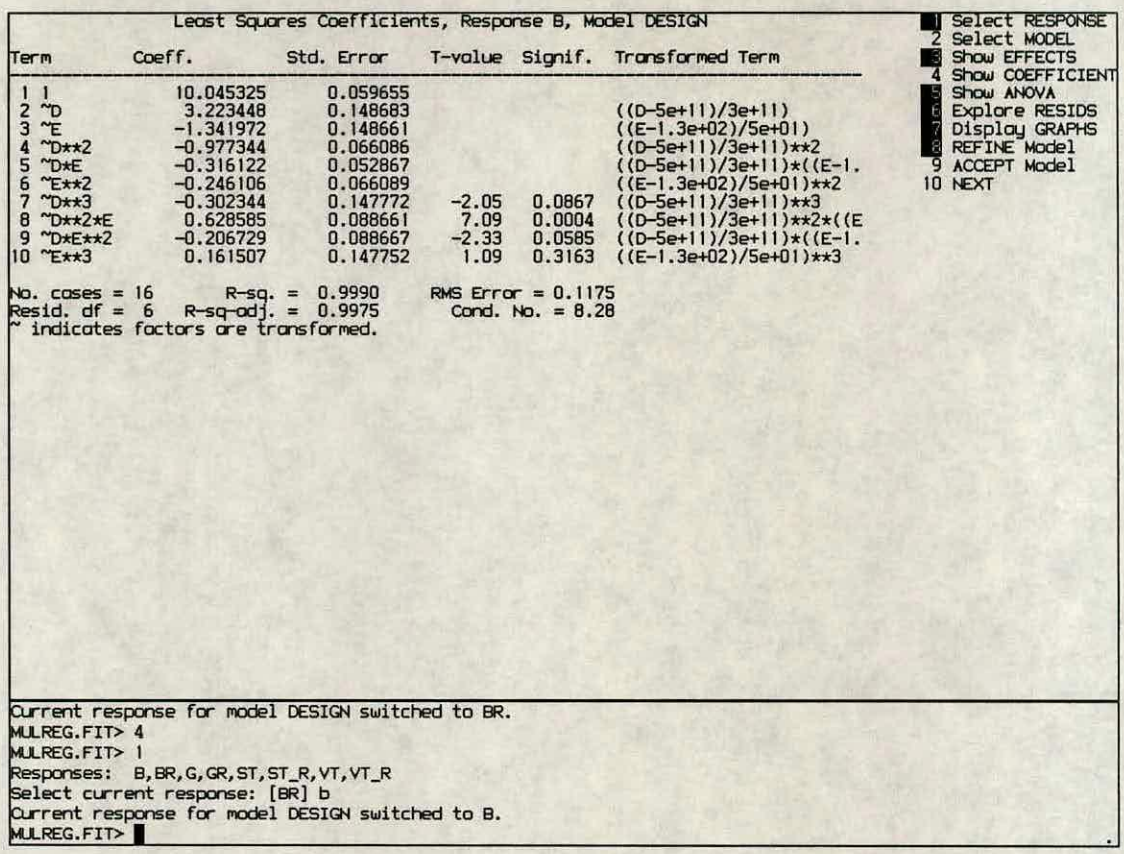


Figure 4–16: An example of a coefficients table.

significance value and vice versa. If the T-value is low (a high significance) then the term can be considered as contributing little to the overall fit.

If the factors are orthogonally coded, ie they are all in the range $-1 \leq 0 \leq 1$, then the coefficients can also be used to analyse the effects of the different model terms.

An example of a coefficients table is illustrated in figure 4–16.

Analysis of variance table The analysis of variance table can be used to determine if the model can adequately account for the variation in the data. Essentially it separates the variability in the data into contributions by each of the factors and the error. The assessment of the fit is performed using an F-test (and its

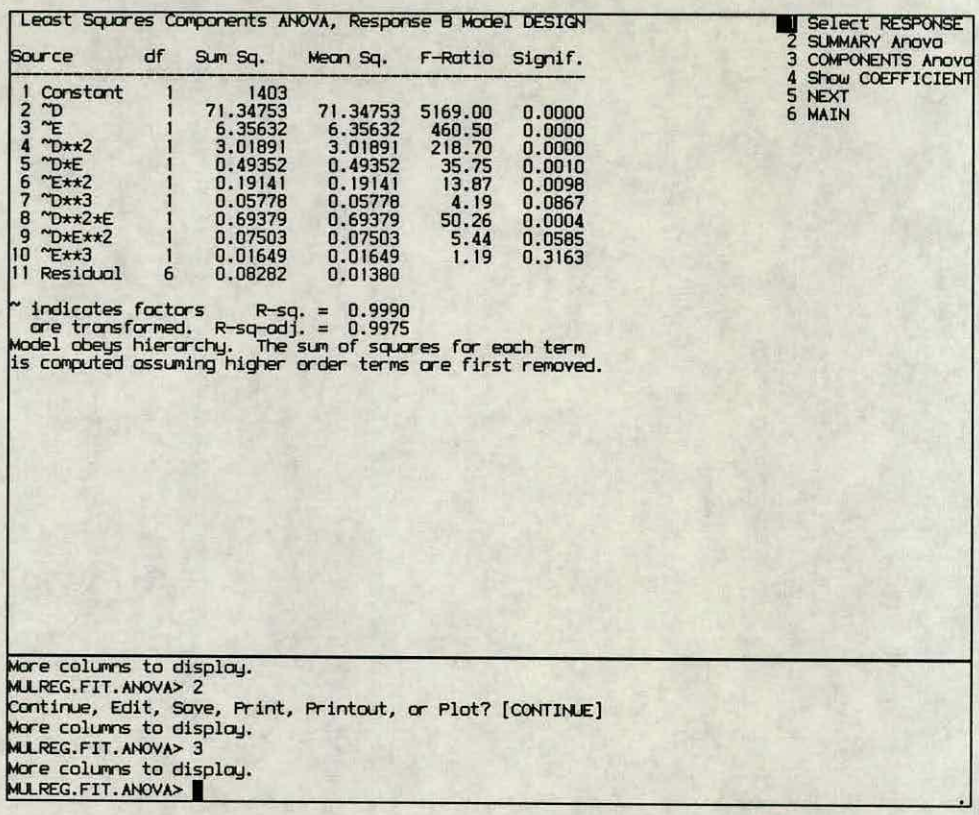


Figure 4–17: An example of an ANOVA table.

associated significance) and the value of R-squared. These are described in more detail in [46] but will be discussed briefly below.

The value of the F-test is a measure of the fit. It is used to test whether all the coefficients can be considered zero. Its associated significance value is the probability of observing an F value as high or higher than the one observed, if all coefficients are considered zero. If the significance value for a factor is low (less than 0.05) then that factor can be considered to have a significant effect on the value of the response and hence is required in the model.

R-squared is also an indication of the agreement between the model and the data. It is defined as the ratio of the predicted sum of squares to the observed sum of squares. A perfect fit would correspond to a R-squared value of 1 or 100%.

An example of an ANOVA table used to describe the different factors and their corresponding F-values is shown in figure 4–17.

RPL

RPL is the RS/1 programming language and can be used to:

- Perform multiple RS/1 commands
- Perform repetitive tasks
- Tailor RS/1 for particular applications

The language is particularly useful for handling and structuring large amounts of data in eg tables. There are essentially three types of RPL procedures:

1. **Public procedures** These are available to all and are integral with the program.
2. **User procedures** These are private to the owner of the procedure itself.
3. **Group procedures** These are user-written procedures which can be shared within a group, who would all require access to the same user-written procedure.

An example of a simple user RPL procedure is detailed below. This procedure reads the data from the experiment, `EX_NAME` and writes it to a system file called `FILENAME`. This routine was used to store the data from the experiment worksheet to an operating system file, which could then be used to generate the required simulation files from the conditions specified by RS/1.

```
procedure(EX_NAME,FILENAME);
    designtbl = cat(EX_NAME,"@worksheet");
    if tableexists(designtbl) then
       begin;
         call public $ez_writefile(designtbl,FILENAME,1,
              lastrow(designtbl),"|",false,empty,empty,empty,true);
       end;
```

```
    else
        type "design table does not exist";
end;
```

4.4 Conclusions

This chapter has sought to outline the basic theory behind experimental design and the different techniques available to the user. One of the most powerful techniques is Response Surface Methodology. This essentially uses standard statistical designs to design an experiment, usually with a reduced number of runs than a normal full-factorial design. A model is then fitted to the measured or simulated data at these different conditions. From this model much information can be gained concerning the variation of the output responses to the control factors.

Another technique that has been used in the design of experiments is Taguchi analysis. This is particularly concerned with the design of quality into a product at the beginning of its cycle. It considers how particular responses vary with slight noise variations in some of the input factors, using signal-to-noise ratio analysis and level (or mean) analysis as important tools in the design.

A simple example has been shown for both RSM and Taguchi analysis. This has sought to establish the relevance of these techniques to the semiconductor industry and the benefits that can be obtained from these methods. A particularly useful tool in the design of experiments or design for manufacturability is RS/1, the statistical package from BBN Software Corporation. This provides an entire framework in which to design and then comprehensively analyse the results of the experiment.

Chapter 5

Parallel Computing

5.1 Introduction

Computing and the resources available have greatly progressed from the first commercial computer, UNIVAC 1, in 1951. One person who was involved at these early stages was John von Neumann, who is now universally credited for formalising some of the salient concepts in computer systems. It was he who established the structure known today as the *von Neumann architecture*. This architecture is common to virtually all standard sequential computers built to date. It has, however, a limitation in that bottlenecks can occur in memory/processor data access. The general outline of a von Neumann architecture is illustrated in figure 5–1.

Despite limitations, the von Neumann architecture has remained the dominant force in the majority of applications. This is mainly due to two factors [64].

1. There has been a large amount of time and money invested in software tools and languages for these serial machines. As a result, due to inertia, there has been a reluctance to change from this de facto standard.

2. Due to the rapid increase in conventional computing performance, end users have often decided against any change in the status quo.

In reality, however, the only way to really break out of the mould, so to speak, is to adopt some form of parallel processing or parallel computing techniques. Such an example is the multiprocessor distributed memory parallel machine, since each

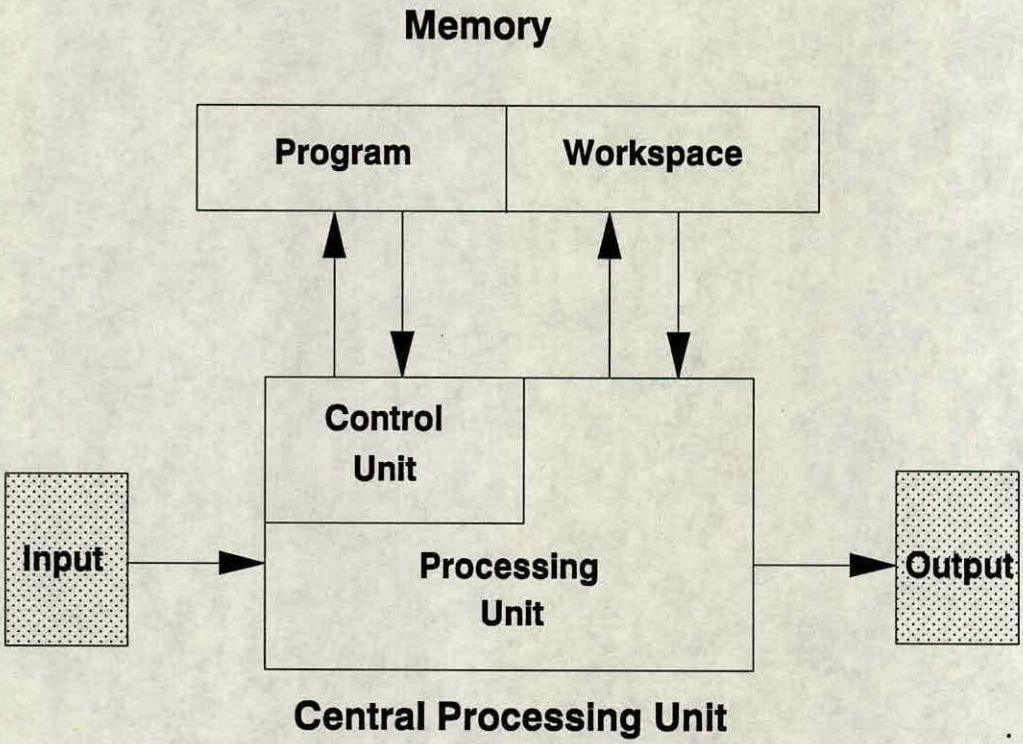


Figure 5–1: A general von Neumann architecture

processor has access to local memory, the problem of the von Neumann bottleneck is removed.

The computing power available at present on many parallel machines enables users to contemplate problems that would otherwise have been unthinkable.

Problems do, however, arise when switching to a parallel solution. There is usually the requirement of having to port the user program onto a parallel platform. This is often easier said than done, because it is not uncommon for the code to be written in an uncommented and unstructured format. Thus there is a struggle to not only overcome the problems inherent in unstructured code, but also the knowledge that needs to be gained on the parallel computing learning curve. As software tools and environments for parallel machines continue to improve the users of parallel computing should move from the relative expert in the field, to those with only a general computing background.

In the following sections some of the available parallel platforms and software

tools will be reviewed, together with a more detailed look at some processors used in multiprocessor, distributed memory machines.

5.2 Classification of parallel computers

As the parallel computing scene has developed over the years, as outlined above in section 5.1, so has the need to categorise the different systems into various classifications. Available categories are by no means complete and often it is a non-trivial task deciding into which grouping or taxonomy a particular system falls. Generally the characterisation methods seek to investigate the number of instructions that can be performed simultaneously, the arrangement of the processors, how they are connected and how data flows through the system. Questions that need to be asked are; is each processor operating on the same or different instructions? Does the system utilise shared or distributed memory? Do any of the processors use vector processing or pipelining? For a more detailed analysis of system classifications see [65-68].

The most commonly used taxonomy at present is that proposed by Flynn [69] in 1972. Flynn did not describe the system in question with respect to its architecture or structure, but rather to the instructions and operands being passed around the system. He considered the instructions as an *instruction stream* from the memory to the processor and the operands as a *data stream* to and from the processor. Based on this, four main classifications were established.

- SISD - Single Instruction Single Data stream.
- SIMD - Single Instruction Multiple Data stream.
- MISD - Multiple Instruction Single Data stream.
- MIMD - Multiple Instruction Multiple Data stream.

Figure 5–2 illustrates the different Flynn taxonomies.

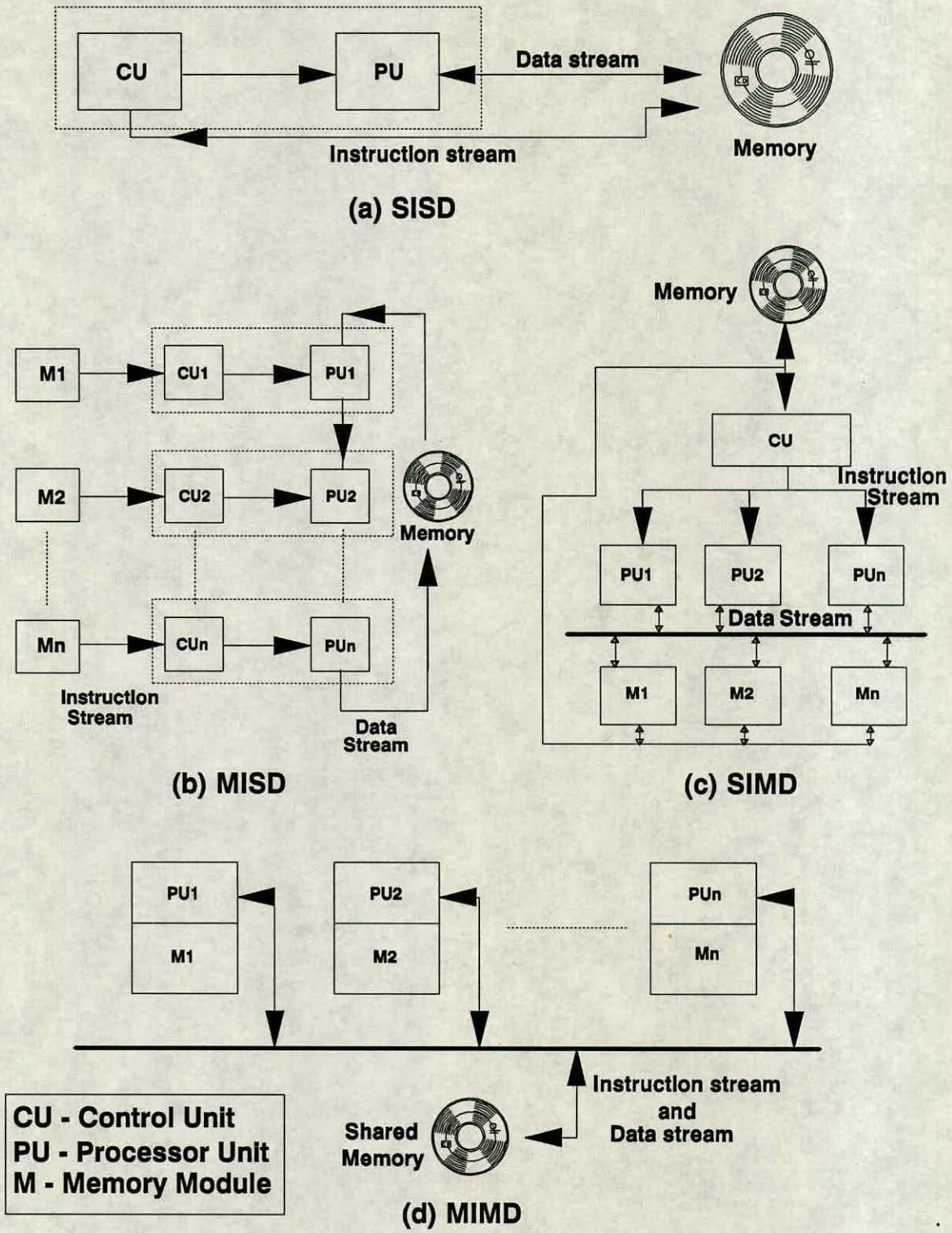


Figure 5–2: Different Flynn Taxonomies

5.2.1 SISD

This is the standard sequential machine, possessing the von Neumann architecture. There is a single stream of instructions and each instruction leads to one operation, thus resulting in a single data stream. According to [66] there are four types of parallelism in SISD systems. These are:

1. Multiprogramming

This is prevalent in large single processor systems where several programs are resident in memory at the same time and each program or user has a share of processor time, known as *time sharing.* This is particularly efficient when the CPU is idle, thus allowing another user program to be executed.

2. Spooling

This involves the use of a spooling program. This program is responsible for control of printing routines, thus leaving the processor free to continue execution.

3. Multi-function Processor

As the name suggests, this form of architecture possesses a number of processing units, but still within the one control unit. As a result different data can be operated on in parallel.

4. Pipeline operation

This uses the principle of different instructions being executed in parallel eg fetch and execute commands.

Examples of SISD systems are many. The most ubiquitous being the humble IBM PC or compatible, the general class of SUN workstations and the earlier Cray-X series supercomputers, and the Control Data Corporation (CDC) 6600 and 7600 [70].

5.2.2 SIMD

This structure has a large number of processing elements with a single controller and only one copy of the user program. The controller executes a single instruction at a time to all of the processing elements which execute the instruction on their own local data set. There are usually extensions to standard languages to facilitate their use on such machines, eg transfer of data between processors.

One of the most common examples of a SIMD system is the Advanced Memory Technology (AMT) Digital Array Processor (DAP) [71]. This machine consists of an array of 32 x 32 or 64 x 64 processing elements, with a clock speed of between 80 - 100 ns. The machine at Edinburgh University contains 4096 processors in a 64 x 64 grid and is hosted from a Sun workstation. The language used is called Fortran-plus, which is a derivative of Fortran.

Another example of an SIMD system is Thinking Machines' Connection Machine CM-200 [72, 73]. The Connection Machine, often referred to as a data parallel machine, provides an integrated system of hardware and software. The hardware includes front-end computers for code development and up to 64K processors for data parallel operations. Software elements consist of UNIX and a programming environment for parallel applications. The languages supported are Fortran 90, C* (an extension of C for data parallel programming) and *Lisp (an extension of Common Lisp).

Typical applications for SIMD machines include image processing, where pixels can be farmed out to the different processors, with neighbouring image sections being on neighbouring processors, thus communications are only with nearest neighbours.

5.2.3 MISD

These machines would involve multiple instructions operating on a single data set. There are some supercomputers, including the Cray-1, where it has been suggested

that they might come under such a classification [67]. Strictly speaking, however, no present computers can be classified in this respect.

5.2.4 MIMD

This classification allows general purpose high-performance computing using multiple independent processing units. These units execute multiple instructions on multiple data, thus each processing element can execute the same or different programs.

Generally speaking, these machines can be divided further into shared memory machines and distributed memory machines. Figure 5–3 illustrates the differences between shared and distributed memory.

Shared Memory

This classification is also known as a *tightly-coupled* system. Broadly speaking this will consist of two or more processors, with each processor having a link to a shared or global memory. The advantage of this structure is that in general it is relatively simple to program. The great drawback is, however, that there is a finite limit to the number of processors that can be added. This is due to the processor to memory bottleneck that occurs, when the number of processors increases beyond the optimum limit. To overcome this problem local memory caches have been introduced, this is, however, the beginnings of a distributed memory architecture.

Examples of shared memory MIMD machines include the Cray Y-MP [70] which has 8 vector processors and parallelism is provided by the compilers. Cray's new vector machine the Y-MP C90 also comes into this category. This is a full 16 cpu system and is reputed to be four times faster than the Y-MP, boasting a peak performance of 16 Gflops, where a flop is a floating point operation per second [74].

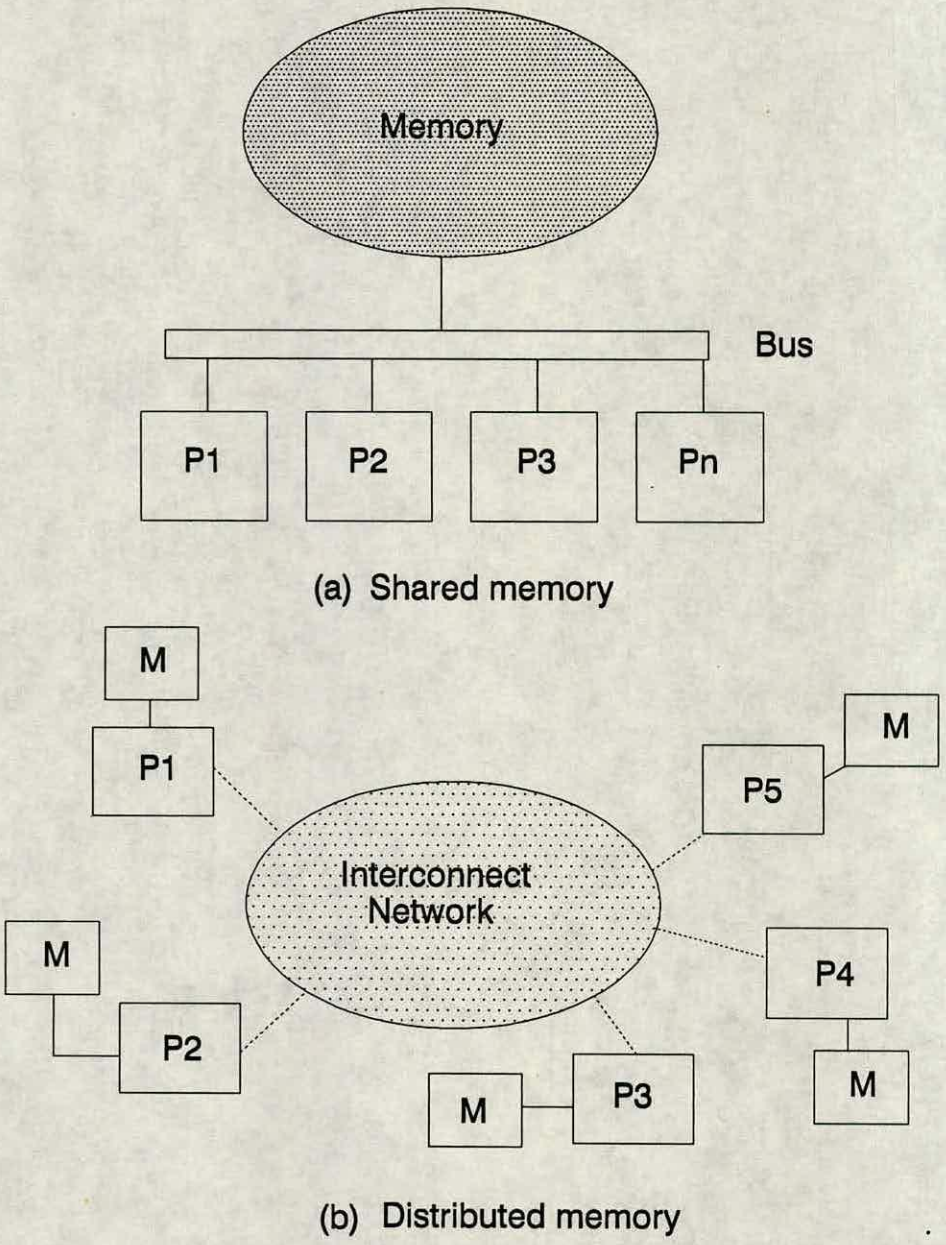


Figure 5–3: General architectures of (a) shared memory and (b) distributed memory systems

Distributed Memory

This is also termed a *loosely-coupled* system. As illustrated in figure 5–3, this type of architecture has local memory available to each processor, thus adding flexibility, both in terms of connection of processors and addition of extra processors. This flexibility, however, also introduces problems and questions that need to be asked; How will the processors be connected together? How will they communicate with each other? Utilising a bus structure only leads to the bottlenecks of shared memory and it is not practical to connect each processor to all the others. Thus each processor is connected to only a few others on the total system. This connection or topology can be fixed as is the case for hypercube machines [75] or user flexible topologies as offered by the Meiko Computing Surface [76]. Examples of some possible topologies for a transputer network are outlined in figure 5–4.

These systems generally use a message passing system to send and communicate to other processors on the system. For more detail regarding message passing systems see section 5.3.

Examples of MIMD machines include offerings from Meiko and their Computing Surface. The processing nodes are traditionally transputer or i860 Intel chips. Recently, however, Meiko announced the availability of its latest MIMD machine called the Relational Datacache (RDC). This consists of 40 MHZ SPARC processors which run an implementation of the standard ORACLE 6.2 relational database system, above SunOS Unix. Two T800 transputers are used to control communications.

Although the categories outlined above give a helpful coarse division of the large diversity of systems available at present, in reality some architectures exhibit aspects of more than one category. For example the Cray X-MP can be considered to be both MIMD (multiple processors) and SIMD (pipelining for vectorisation). Where there are multiple processors the memory can be shared or distributed or both. There may or may not be caches or virtual memory systems. Interconnections may be achieved using methods such as crossbar switches or multiple bus connected systems.

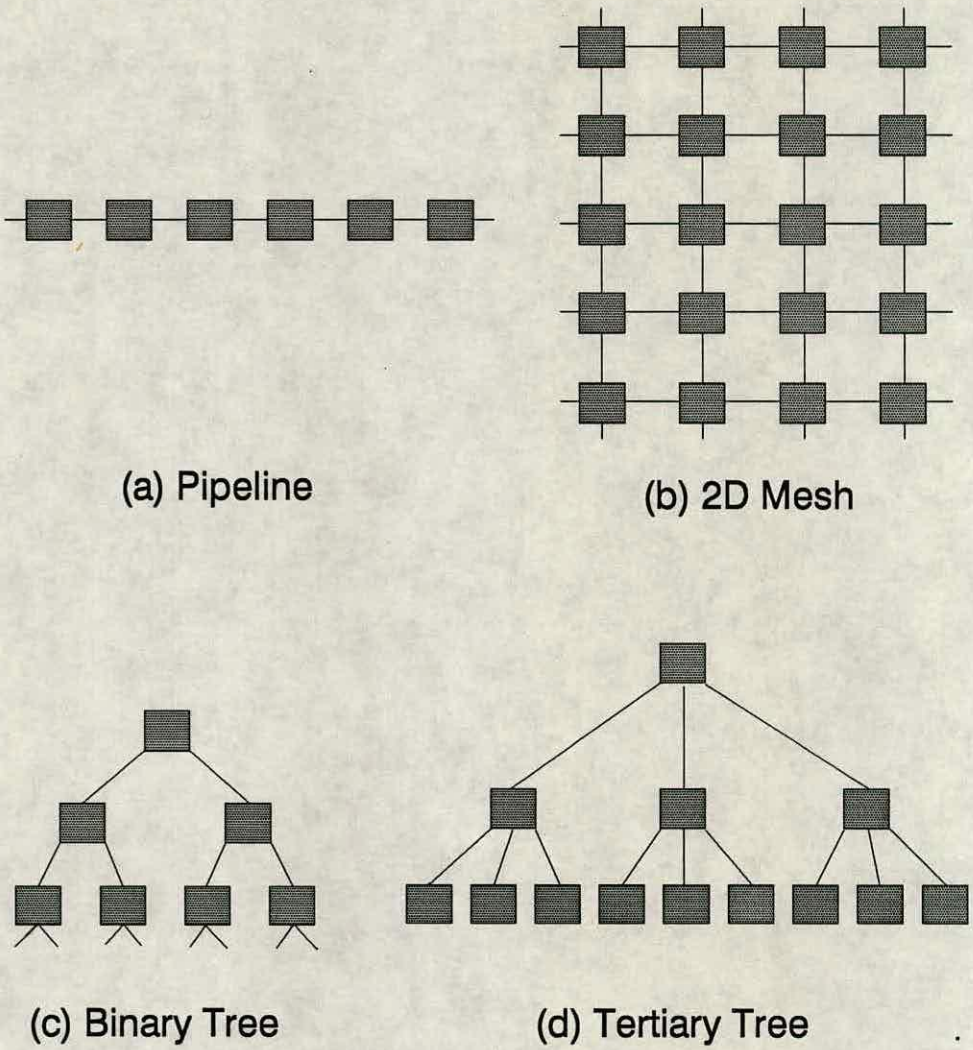


Figure 5–4: Examples of ring, pipeline, binary tree and tertiary tree topologies

Thus in conclusion, not all machines can be neatly classified into one of Flynn's four taxonomies, but to date it is probably the most useful and common of the classification systems available and provides the user with a general insight into the type of machine in question.

5.3 Message Passing Systems

Due to the nature of MIMD programming, it is essential to have some form of message passing system for inter-processor communication. The greatest problems arise when messages are to be sent to remote processors, thus the messages need to be routed through a number of other processors first. Other aspects that need to be considered are deadlock avoidance, ie two processors waiting for each other for communications, and efficiency, ie ensuring that the processors spend as little time as possible idle.

A number of the more widely used systems that include message passing are outlined below.

5.3.1 Tiny

Tiny is a message passing system developed at the Edinburgh Parallel Computing Centre [77]. It consists of a library of procedures that can be called to send and receive messages around the system. These can be called either from Occam or from the 3L Toolset utilities. Tiny will be outlined in more detail in a working system in section 6.3.2.

5.3.2 CS Tools

Communicating Sequential Tools was developed by Meiko as a development suite of software for parallel applications [78]. One of the advantages of CS Tools is that development of code takes place at the existing user environment, thus the programmer does not need to learn a new environment. As the name suggests, CS Tools provides the toolkit to enable the user to exploit the full potential of the parallel machine. Essentially it works on the principle of the programmer executing different program parts simultaneously on different processors.

Communication takes place using a Computing Surface Network (CSN). This allows the user to create a *transport* for a process and register the name with the CSN. The CSN can then be used to pass messages between processes that have registered transport names.

The use of some of CS Tools facilities in a real situation is outlined in section 6.4.1.

5.3.3 Linda

Linda was first developed by David Gelernter [79]. It approaches the communication problem by use of a shared memory model. This acts as a store for messages sent and can be used to split up data structures between processors very effectively. If these structures are split up into a number of parts, then the traditional bottleneck of shared memory systems can be avoided. Although traditionally used on shared memory machines, Linda can also be used to emulate shared memory by message passing.

5.3.4 Strand

Strand, or Strand 88 as it is usually referred to, is a highly portable and scalable parallel programming language, designed for execution on a variety of hardware platforms [80]. These platforms include the Meiko Computing Surface, Sun SPARC station and general transputer based systems. The nature of the language is that much of the parallelism is expressed in the software, thus reducing the knowledge requirements about the hardware configuration. This results in highly portable code. The programming is carried out in Strand's own language, but has interfaces to C and Fortran.

5.3.5 Express

Express is the name given to a particular range of products used in MIMD machines and is marketed by ParaSoft Corporation [81]. The Express Communica-

tions Environment allows for synchronous and asynchronous inter-processor communications. Each processor is given a separate numeric id. The popularity of Express has been gained from the fact that it is presently available on a wide range of hardware platforms, including the Meiko Computing Surface and hypercubes.

5.3.6 Helios

Helios is a multi-tasking UNIX-like operating system developed by Perihelion Software [82]. The design is based on the client-server model, where user tasks request services from system provided server tasks. The Helios system is thus present on each processor on the machine, to enable service requests to be implemented. There are essentially three ways in which to develop programs on Helios. The first is just as a sequential model, running under the Helios operating system, as if under MS-DOS or UNIX. The second is the running of sequential tasks, communicating through pipes. Instead of time sharing this operation, Helios can carry out the tasks in parallel on separate processors. The final approach is using "traditional" parallel programming techniques, where the program is divided into a number of separate tasks and distributed across the network.

5.4 Software Design and Parallel Algorithms

The problems of designing new software on a sequential machine are many, however, these problems are multiplied when considering parallel machines. Aspects of parallelism that need to be considered are:

- Architecture
- Communication
- Decomposition
- Granularity

- Load Balancing

These will briefly be outlined in more detail.

Architecture It is important to choose the best machine architecture for the task in hand. Considerations must include the memory requirements and processing capability of each processor. If restricted to one type of parallel machine, this will often dictate the type of parallel algorithm chosen (see section 5.4.1).

Communication It is essential in any design to reduce the inter-processor communication between processors. This is also a hardware consideration, as the chosen topology can reduce communication overheads greatly.

Decomposition This is the decomposition of the program into different smaller parts to allow parallel execution. A good example of this is image processing analysis, where different sections of the screen image can be executed by different processors.

Another example of decomposition can be illustrated by the following Fortran example program:

```
Do I = 1, 20000
Call Function (I)
End do
```

This could be implemented on a number of different processors by splitting up the array `(I)` into a smaller subset and each processor operating on that subset in parallel.

Granularity This is essentially a trade off in choosing the correct degree of parallelism. If a problem has undergone decomposition, is it best to utilise all the available processors in working on the problem. Often if the code is decomposed

into very large numbers of sub-problems, then the communications overhead becomes greater than the computational advantage to be gained. Thus granularity is really the degree of decomposition and to obtain maximum performance from the system, this must be optimised wherever possible.

Load Balancing This addresses the problem of execution times for the sub-programs resulting from decomposition. Since the total execution time is dependent upon the time taken for the largest task, it is often more efficient to utilise the other processors' inactivity while the longest task is finishing. The ideal scenario would be all processors finishing at the same time.

5.4.1 Parallel Algorithms

These are algorithms that are used to broadly describe the different forms of parallelism exploited at present [83]. They are termed *geometric parallelism*, *algorithmic parallelism* and *event parallelism.*

Geometric Parallelism This is also commonly known as data parallelism or domain decomposition. The problem is divided into subsets, with each processor operating on a particular subset. The geometric term refers to the fact that neighbouring processors execute on the data from neighbouring data sets. As a result the only communication required during computation is the passing of boundary values. The communication load is proportional to the size of the boundary of the data set and the computational load is proportional to the actual size of the data set itself. Examples of this type of parallelism are hydrodynamics, image processing and process simulation [84].

Algorithmic Parallelism This is also referred to as data flow parallelism, because the configuration is usually in the form of a pipeline and the data flows along the pipeline. The algorithm is broken up into different parts which are assigned to different processors. The communication load on each processor can be

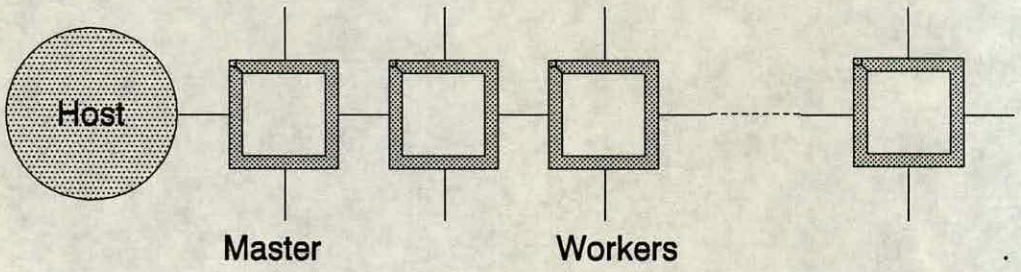


Figure 5–5: An example of a simple taskfarm or event parallelism

greatly increased with this type of parallelism and without care, communication bandwidth problems can arise. Often the data space required for each processor is small and if using a transputer array, only the internal memory is required.

Event Parallelism This is also known as the taskfarm. It can be considered as a master transputer allocating a number of tasks to worker or slave processors, where each worker usually executes the same program, but on a different data set. As a result the different worker tasks are totally independent of each other and interprocessor communications are kept to a minimum. A typical taskfarm arrangement is outlined in figure 5–5.

Bottleneck problems can occur, however, with a large taskfarm especially in the form of a pipeline. This is because all the commands to and from the master must be routed through worker number 1 (see figure 5–6(a)). This problem can be alleviated in the case of a tree structure (see figure 5–6(b)). In this scenario messages are routed to the master through workers 1, 2, and 3, thus easing the bottleneck.

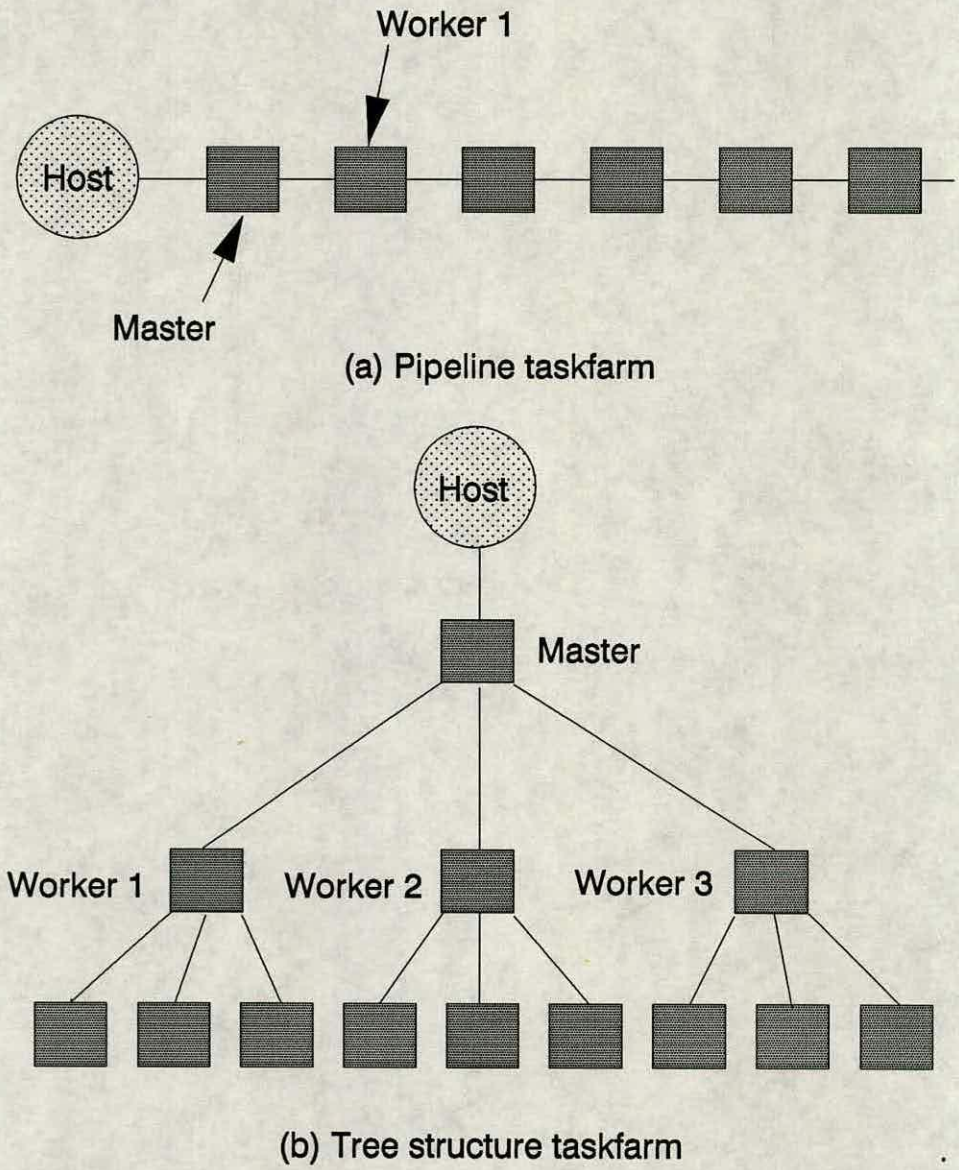


Figure 5–6: A comparison of a pipeline taskfarm (a) and a tree structure taskfarm (b)

5.5 The transputer

5.5.1 Introduction

The transputer, a *trans*(istor-like com)*puter*, which has popularly become known as "the computer on a chip", was first developed by Inmos Ltd in 1984. With the problems associated with the Von Neumann bottleneck, the transputer was one of the first chips to really address the issues of parallel processing. The transputer is unique in that it is a building block, while it contains its own memory and processing elements. It also features serial links which allow it to communicate with other transputers [85, 86]

There are three basic types of transputer manufactured by Inmos (now SGS-Thomson Microelectronics): the 32 bit T414, the 16 bit T212 and the 32 bit T800 with a floating point unit. These transputers and the T800 in particular, have been the building blocks of many large distributed MIMD computers in operation today. The latest offering from SGS-Thomson, the T9000, is likely to be used even more as a future building block. For more details of the T9000 see section 5–8.

5.5.2 Architecture and hardware

As already detailed the transputer in its basic form contains a processor, memory and a number of standard point-to-point communications links, all on the one piece of silicon. It may be regarded as a small *reduced instruction set computer* or RISC as it only possesses a small number of instructions.

The T800 architecture is illustrated in figure 5–7.

The real novelty behind the transputer is the communication links. These links, since they are on chip, result in maximum speed of operation and minimal wiring. Each link operates at a rate of 10 megabits per second and since the four links can operate simultaneously, this results in a total throughput of 40 megabits per second. The links also facilitate the construction of large networks of transputers

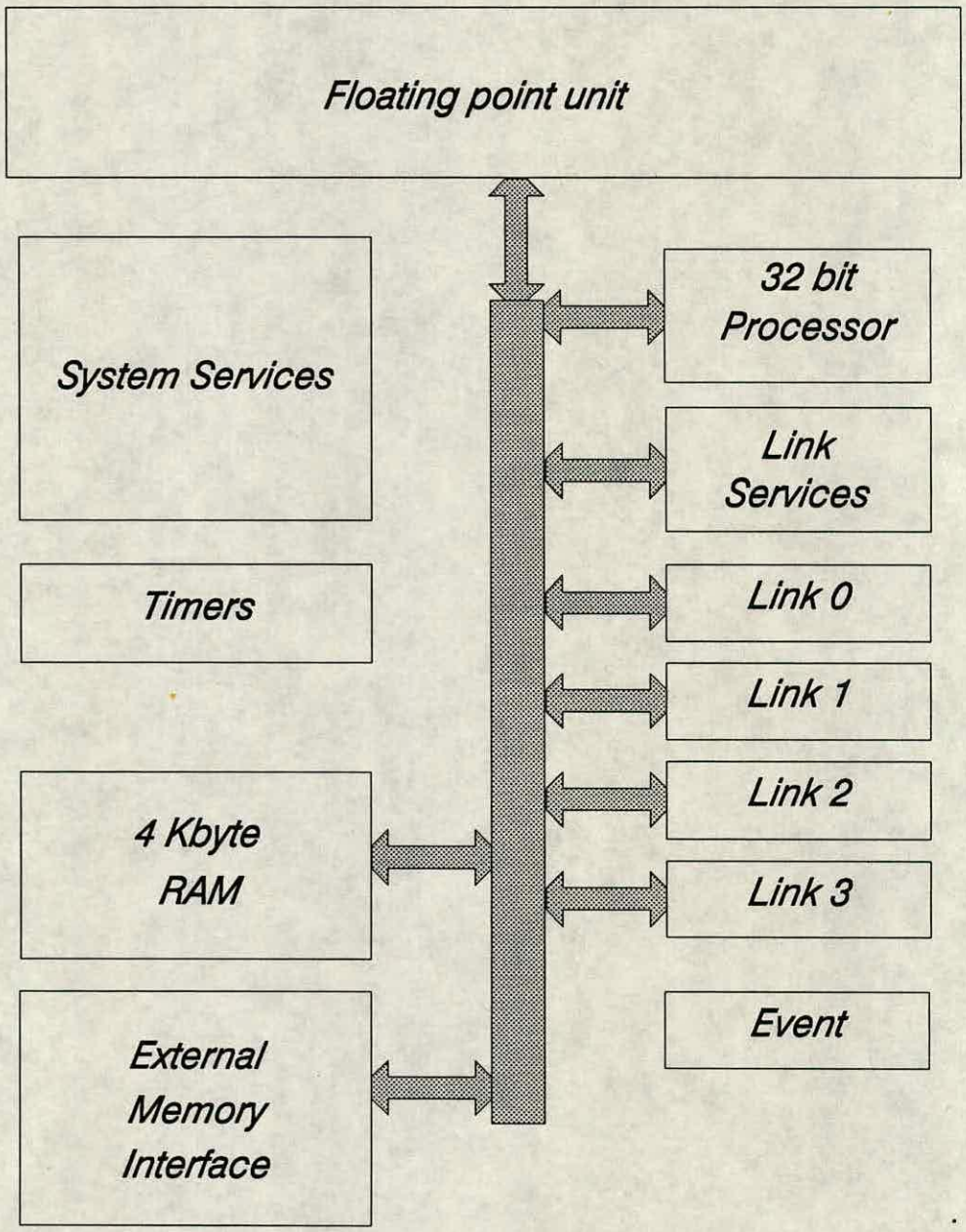


Figure 5–7: The T800 architecture

	T212	T414	T800	T9000
Processor	16 bit	32 bit	32 bit	32 bit
Floating point	software	software	hardware	hardware
Clock	20 MHz	20 MHz	30 MHz	50 MHz
On chip RAM	2 KByte	2 KByte	4 KByte	16 KByte
MIPS (peak/sustain)	20/10	20/10	30/15	200/70
MFLOPS (peak/sustain)	–	0.1	4.9/2.25	25/15

Table 5–1: Table showing a comparison between the members of the transputer family

with arbitrary topology eg mesh, pipeline, tree. Flexibility is inherent in the design and further transputers can easily be added to the network.

When considering the performance of the transputer it is best to compare the different members of the family. Table 5–1 illustrates the relative differences between the T212, T414, T800 and the future T9000.

As a result of the flexibility offered by this computer on a chip, the transputer finds itself being used for a variety of new machines and in a large diversity of applications. As the software available for the transputer improves and as the T9000 will soon be available, the future augurs well for the workhorse of parallel processing. These two aspects; software and the T9000, will now be considered briefly in the next two sections.

5.5.3 Software tools

One of the early problems with the development of the transputer was not any inadequacies in the hardware or an implementation of it, but in the software tools that enable the user to tap the full potential offered by such a device. Over the last few years, however, much software has become available, in the form of operating systems, message passing systems, languages, compilers and debugging environments. It is outside the scope of this work to give a full review of all such

software, but some of the major packages for the transputer will be outlined below, together with those already detailed in section 5.3.

The Occam language has become synonymous with the transputer and has been called the assembly language of the transputer [87]. It was developed by Inmos for concurrent systems, where tasks, or processes exist as self contained units and communication is performed by channels. It has the advantage that it can easily utilise the parallelism and speed up offered by the transputer. On the other hand, however, it has the dual problem of the programmer having to learn a new language, but perhaps more importantly, much of the existing application programs are written in C and Fortran, thus requiring the rewriting of code into Occam. Because of this, the Occam language, although popular in some circles, has not really been adopted by the majority of transputer users.

One of the most popular suite of packages available for the transputer, is the 3L Toolset [88]. This comprises of a number of different language compilers and programs available historically for the IBM PC with a rack of transputers. The packages available include Parallel C, Parallel C++, Parallel Fortran, Parallel Pascal and Tbug, a source level debugger. The use of the 3L Toolset and its constituent programs will be outlined in more detail in section 6.3.

Developments in operating systems have led to IDRIS [89], which was released for the Parsys family of transputer based high-performance computers. It is a UNIX compatible operating environment and conforms to the POSIX standard. It consists of a multi-user, multi-tasking executive along with a set of utilities, (including X-windows) many of which have been developed for the Parsys SuperNode 1000 series of computers. Each transputer has a full operating system kernel, with communications taking place using standard UNIX sockets. To prove the portability of the software, the Oracle database program has recently been ported to the IDRIS system, something which may be the kick start required for other commercial programs to be implemented on parallel systems.

Although this section and section 5.3 has only scratched the surface of available software, it is hoped that it has given a flavour of the type of tools available on

the market at present. After a slow and somewhat shaky start such software is now beginning to utilise the parallel machines to their full potential.

5.5.4 T9000

The T9000 is the name given to the latest transputer soon to be available from SGS-Thomson. There were four basic design goals set for the T9000 [90]

- An order of magnitude increase in performance over the T800.
- Upward compatibility with the previous series of transputers.
- Enhanced support for operating systems in embedded applications.
- Enhanced support for multiprocessing via improved interprocessor communications.

These goals have been broadly met in the information available to date from SGS-Thomson, although the real test will be when the silicon arrives with the end user. The T9000 will have a peak performance of 200 MIPs and 25 MFlops, and together with a dedicated on-chip communications processor and high speed links, it will offer vast multiprocessing power. Figure 5–8 illustrates the general architecture of the T9000.

Also developed along with the T9000 is the C104 chip. This chip behaves in a similar manner to a telephone exchange and enables the T9000's to be connected together. Each C104 router can connect 32 transputers, all four links of each transputer can be connected to the same or different routing chips. The C104 looks at the leader of each packet arriving on a link input and uses this to determine the output link for that packet. The virtual channel processor on each transputer is responsible for mapping a large number of communications channels onto the four physical links and passing messages on to the C104. As a result message passing congestion should be greatly decreased.

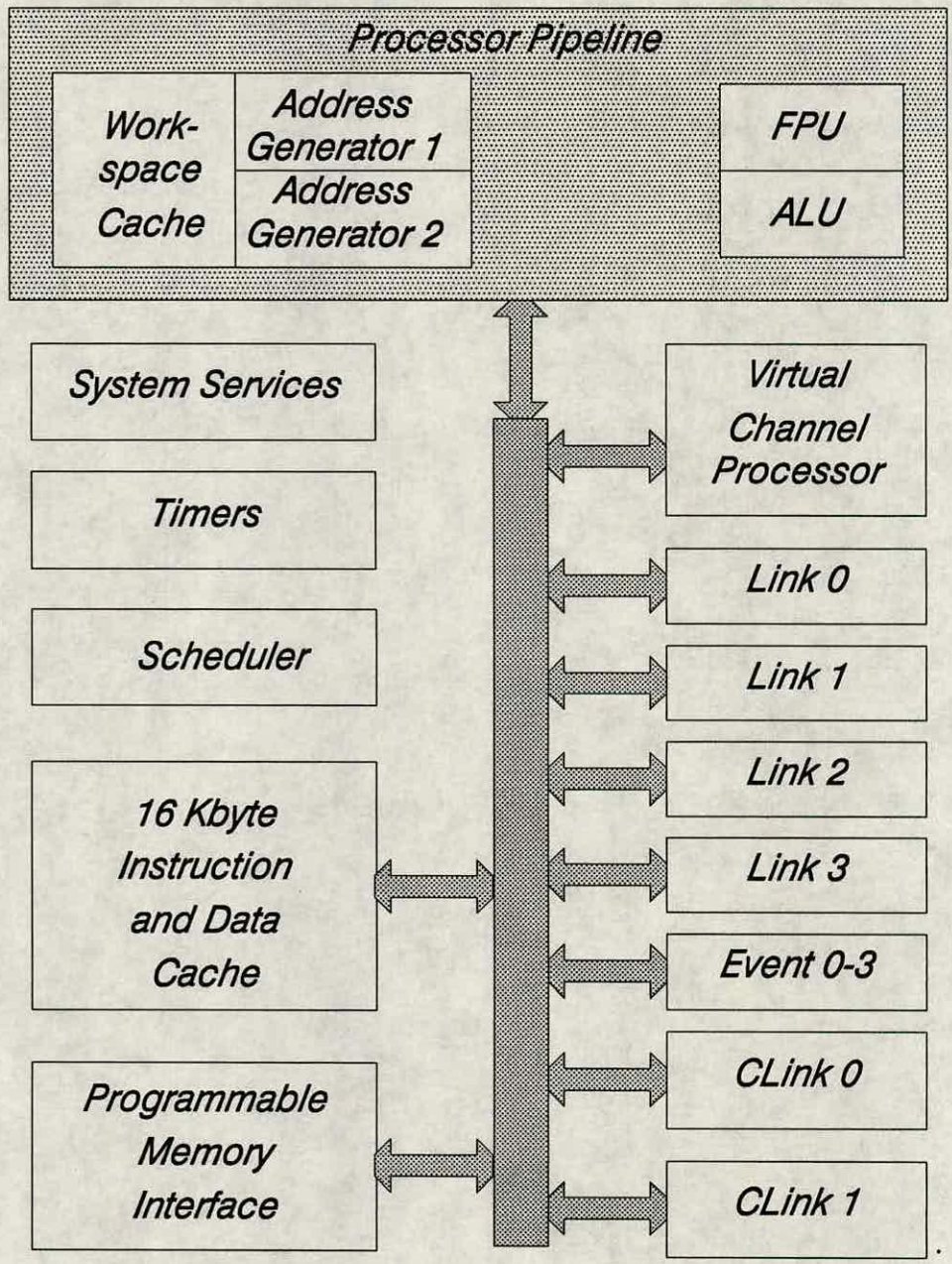


Figure 5–8: The general architecture of the T9000

The T9000 thus offers not only a high performance processor, but also an effective communications strategy. A programming model can now be mapped directly onto a set of virtual channels routed through a C104. There are also no longer any more restrictions to the number or topology of transputers that can be connected together. With this impressive package soon to be available and given the track record of the T800 and T414, the future augurs well for the transputer.

5.6 Intel i860 - A brief overview

On February 27 1989, Intel Corporation announced the first ever 1 million transistor microprocessor [91, 92]. The i860 XR, as it was called, approached Cray 1 performance on a single chip. The main constituent components of the chip are as follows and are illustrated in figure 5–9.

- A RISC integer unit.
- A floating-point control unit.
- A floating-point multiplier unit.
- A floating-point adder unit.
- A 3D graphics unit.
- A memory management unit (MMU).
- A 4 Kbyte instruction cache.
- An 8-Kbyte data cache.

The RISC architecture means a high throughput of instructions per second and under certain conditions the i860 can perform three operations per clock cycle. One third of the processor is devoted to floating-point calculations with a peak operation of 80 MFlops for single precision data at 40MHz and 60MFlops for

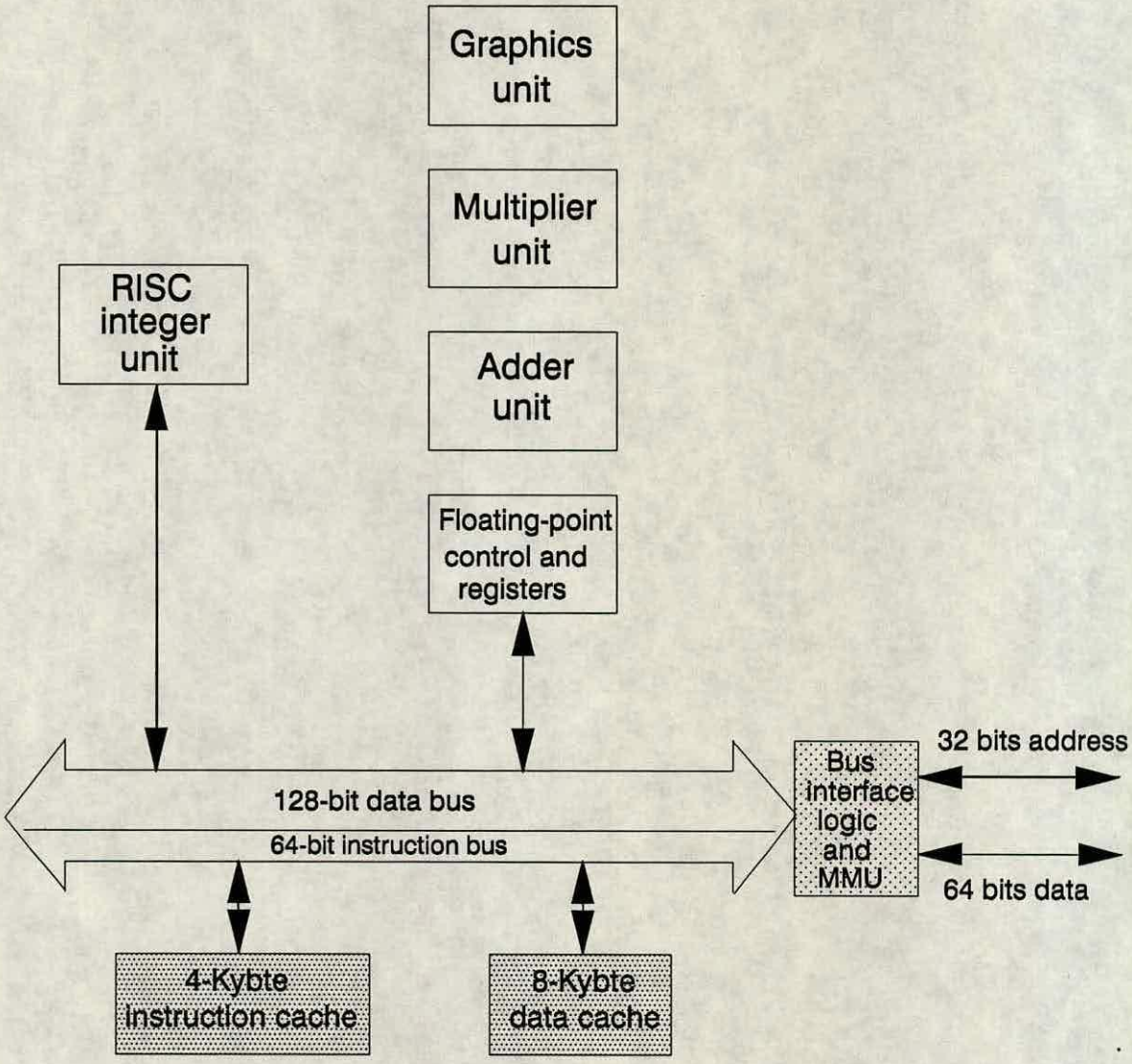


Figure 5–9: The general architecture of the i860 XR

the equivalent double precision. As a result of such high performance the i860 is being utilised more and more in MIMD architecture computers and in embedded applications. One novel application for the i860 is in the PARAstation marketed by Transtech [93]. This consists of up to 4 i860 computing nodes which can be connected to a variety of workstations and PC's. Also available from Meiko is a Computing Surface with i860 nodes, with T800's used to control communications [94].

Software Tools Software support for the i860 has been provided by both internal development teams and independent software experts. The general tools available are C and Fortran vectorising compilers, assemblers, linkers, debuggers and various libraries. Available from Transtech is the i860 toolset [93]. This includes C and Fortran supercompilers, a symbolic debugger, a maths library, signal processing routines and support for image processing.

5.7 Parallel Computing at the Edinburgh Parallel Computing Centre (EPCC)

Edinburgh has established itself as a worldwide centre of excellence in parallel computing. The procurement of parallel machines was originally driven by the Physics Department, but there is now a multi-disciplinary number of users who benefit from the facilities offered.

At present the major systems at the EPCC include:

- A **Meiko Computing Surface** containing over 400 T800 transputers and 1.8 Gbytes of memory.
- A 4-K processor **AMT DAP 608**
- A 1-K processor **DAP 510**
- An **i860 Computing Surface** containing 64 Intel i860's delivering 56 Gflops peak performance and 1 Gbyte of memory.
- A 16K processor **CM-200 Connection Machine** from Thinking Machines Corporation. This has a peak performance of 8 Gflops with 0.5 Gbytes of memory and a 10 Gbyte data vault.

The major funding for the Centre comes from industry and from a joint DTI/SERC Parallel Applications Programme. An annual Project Directory sum-

marises around 100 projects, which are ever growing as more and more users take advantage of the systems available.

5.8 Future trends in parallel computing

As future parallel computers continue to exhibit increased computational power, problems which were previously discarded as impractical will become feasible. This is mainly due to advances in VLSI technology and also in available software tools and algorithms. The feature sizes of devices within the next few years are likely to be as low as 0.1 micron, thus increasing component density and processor speed. According to [95], operating systems will be used more and more in parallel machines and virtual memory management will be commonplace in distributed memory architectures.

Future trends in architecture could mean hundreds of thousands of high performance processors connected together, with improved communication technology easing the increased communication overhead. Also shared memory multiprocessors, each containing about 6 processors, could become an essential element in the construction of powerful parallel machines. The race is on to produce the first Teraflops machine and, at the present escalation in processor performance, this era will only be a matter of a few years away.

One machine which will be capable of delivering 400G flops is the Parsytec GC [96]. This is a scalable distributed memory machine and will use the T9000 as the building block and a parallel extension of UNIX, called Parix, as the supporting software environment. The smallest system available will contain 64 T9000's and the largest will boast 16,834 devices. This is the first of many machines that in the future will benefit from hardware improvements and new specialised parallel software.

At the Supercomputing 92 Show, Cray announced that it hopes to build a parallel supercomputer around Digital's new Alpha processor [97], with the first machine to be available in 1993. The Alpha chips will be integrated into the exist-

ing vector architecture thus allowing both vector and highly parallel approaches to be implemented on the one machine [98]. This is particularly useful as different problems lend themselves to different architectures. This may well be the beginning of a trend in more flexible parallel supercomputers.

In conclusion, the future of parallel computing is secure in the knowledge that we are slowly coming out of our "sequential" way of thinking and realising that a parallel solution is, if not the only solution, often the most cost effective.

As a result of hardware and software advances, no longer will parallel computers only be in the domain of large companies or research establishments, but will become commonplace in many computing establishments throughout the country.

Chapter 6

Process Optimisation using Parallel Computing

6.1 Introduction

The use of the simulation suite of software as outlined in chapters 2 and 3, along with the statistical techniques outlined in chapter 4, is an extremely powerful combination for the optimisation of semiconductor devices, since simulation is typically much faster than fabrication. As a consequence, much more information can be obtained, in a shorter time. Unfortunately 2D and 3D simulations are relatively slow and even with the advent of faster workstations, the time taken to run each of these simulations could quite easily reach 24 hours. On the assumption that other users do not use the system, if 20 different experimental points were chosen then the final results would not be obtained for 20 days. Hence the need for an alternative and faster method of arriving at the final results for the simulations is apparent.

An obvious solution to the problem is to utilise parallel or concurrent processing techniques. These techniques include data parallel methods to spilt each simulation up into smaller units, thus increasing execution time, and event parallelism, where each processor executes a sequential copy of the simulator (see

chapter 5). For the purposes of this work, event parallelism was chosen for two reasons:

1. It was originally envisaged that no access would be given to the source code, (due to its commercial nature) thus making the data parallel approach impossible.

2. An almost linear speedup is achieved for each additional processor added to the system.

Sections 6.3, 6.4, 6.5 and 6.6 detail the major features of different approaches to event parallelism on various platforms and outline the results obtained.

One of the problems in performing large numbers of simulations is the generation of the simulation data files from the design points specified by RS/1. The whole procedure of file generation was automated in a software system to remove the possibility of human error and to increase ease of use and efficiency. This will be outlined in the following section.

6.2 Software system

This section describes the automation of the procedure for generating the necessary simulation drive files from the RS/1 experimental data points and the subsequent extraction of the necessary data from the simulation output or results files. This is also outlined in [99].

Once an experiment has been designed in RS/1, the experimental data points must then be inserted into the different simulation drive files. It is therefore necessary to output the RS/1 data points into an operating system file and then use this file in the generation of the simulation drive files. The general structure of the total system is outlined in figure 6–1.

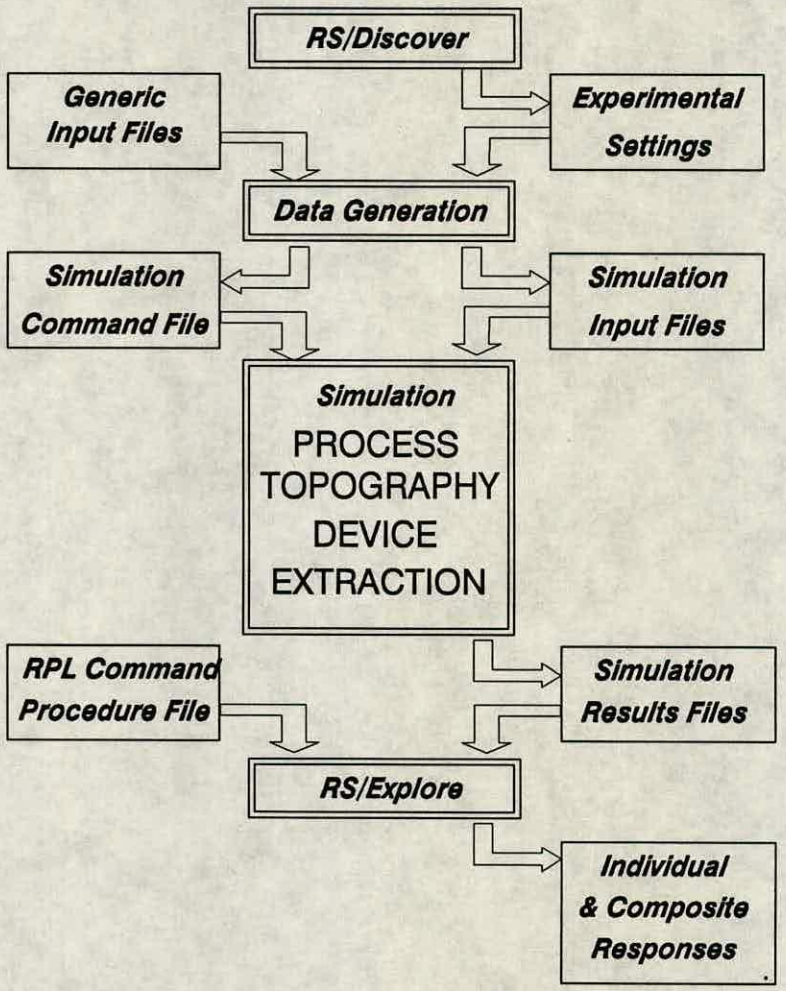


Figure 6–1: An outline of the total software system

The following RS/1 Programming Language (RPL) routine (as described in figure 6–2) was used to generate the operating system file, called the design file, from the RS/1 data points.

A typical format for the design file is shown in figure 6–3. The "|" is used as a delimiter to separate the different data points. In this example there are only two sets of data points, the values of dose and energy for a punchthrough implant.

The design file is used in conjunction with a *generic* file, to generate the necessary simulation drive files. The generic file is essentially the required simulation file with the data values substituted by a pre-defined character or parameter. An example is shown in figure 6–4.

```
PROCEDURE(EX_NAME,FILENAME);
    DESIGNTBL = CAT(EX_NAME,"@WORKSHEET");
    IF TABLEEXISTS(DESIGNTBL) THEN
       BEGIN;
         CALL PUBLIC $EZ_WRITEFILE(DESIGNTBL,FILENAME,1,
              LASTROW(DESIGNTBL),"|",FALSE,EMPTY,EMPTY,EMPTY,TRUE);
       END;
    ELSE
       TYPE "DESIGN TABLE DOES NOT EXIST";
END;
```

Figure 6–2: An example of an RPL routine to generate the design file

```
2e+11|80
2e+11|113
8e+11|113
8e+11|180
8e+11|80
4e+11|80
2e+11|180
8e+11|147
6e+11|180
6e+11|80
2e+11|147
4e+11|180
```

Figure 6–3: The typical format for a design file of two parameters.

```
$ 1.5 um process simulation
$
$EMF's nMOS Silicon Gate VLSI Simulation

$Set up mesh
line x loc=0 spac=.02
line x loc=1.5 spac=.06

line y loc=0 spac=.02
line y loc=.4 spac=.04
line y loc=.6 spac=.5
line y loc=10 spac=10
line y loc=200 spac=200

.
.

$gate oxide
deposit oxide thick=.025

$Boron implant (double implant)
implant boron dose=7e11 energy=25
implant boron dose=?1  energy=?2
.
.
.
plot.2d grid plot.out=ZZZ
```

Figure 6–4: An example of a typical generic file for the SUPREM4 simulator

The simulation file will be exactly the same as the generic file except the ?1 and ?2 parameters (shown for the boron double implant) will be substituted by the required value of dose and energy and the ZZZ parameter in the plot statement will be replaced by the name of the simulation drive file for easy reference on all hard copies of plotted information.

The software to control this procedure was written to allow flexibility of the system and to permit the user to define all the parameters by means of a single control file. An example of part of a control file is detailed in figure 6–5, this defines all the user controllable parameters.

For example the user can specify the number of input factors, the names of the input factors, the number of simulation files to create, the number of responses, etc. The software reads the design file and substitutes the values of dose and energy into individual simulation files, using the generic file as a template. If there are 12 data points in the design file, this would result in 12 different simulation files being created. The program also creates a batch file to run the subsequent simulations. The automation of this procedure not only speeds up the laborious process of creating the necessary simulation files, but also removes any human error which may have occurred in the process.

After the simulations have been performed, routines have also been written to extract the required response data automatically from the simulation results files. This data is then output in the form of the design file, to be read back into RS/1 for statistical analysis, thus resulting in a totally automated system. An example of such a file is shown in figure 6–6. This shows the data points and extracted responses for the input factors outlined in figure 6–3.

It should be noted, however, that it is not always possible to use the extract routines with all parameters, as some responses do require some user interaction.

```
####### Title of experiment: #######
title 1.5 um nMOS Simulation
####### Simulator name: #######
simulator suprem4
####### Design file name: #######
design_name stcnew.dat
####### Generic file name: #######
generic_name rs1gen.dat
####### Prefix for drive file name: #######
prefix run
####### Extension for drive file name: #######
extension .dat
####### No. of drive files to be generated: #######
no_runfiles 12
####### No. of input factors being varied: #######
no_factors 2
####### Names of factors to be varied: #######
factors dose energy
####### Delimiter separating values in design file: #######
delimiter |
####### Variable to search for in generic file ####
search_variable ?
####### Variable to use as a comment in drive file: #######
comment_line $
####### Variable to be replaced by filename in all plots: #######
plot_variable ZZZ
####### No. responses: #######
no_responses 5
####### Names of responses: #######
responses Vpt Vth St Sc gamma
```

Figure 6–5: An example of part of a control file

```
2e+11|80|6.8|0.428|88.3|2.11|0.365
2e+11|113|8.3|0.413|87.8|2.13|0.384
8e+11|113|10.794|0.477|91.3|3.01|0.83
8e+11|180|10.85|0.415|88.2|2.68|0.687
8e+11|80|10.775|0.539|93.6|2.58|0.684
4e+11|80|9.3|0.477|90.9|2.33|0.462
2e+11|180|8.3|0.395|87.3|2.14|0.337
8e+11|147|10.8125|0.432|89.2|2.95|0.761
6e+11|180|10.88|0.41|88|2.47|0.678
6e+11|80|10.8|0.515|92.4|2.42|0.651
2e+11|147|8.9|0.404|87.4|2.14|0.398
4e+11|180|10.9|0.405|87.7|2.28|0.558
```

Figure 6–6: An example of a file showing the input factors and extracted responses for an RS/1 design

6.3 A general transputer taskfarm

NOTE: The work carried out in section 6.3.1 to 6.3.5 was carried out with Dr W.J.C.Alexander.

6.3.1 General taskfarm outline

The following sections outline the salient aspects in the design and operation of the taskfarm. This is also detailed in [100].

Taskfarm requirements

The design of the taskfarm was governed by three major considerations.

- Due to unavailability of the source code, there should be no changes required to the source to facilitate the operation of the system.

- The system should be flexible to use with any user software.

- One file should control all user interaction.

One of the major problems of parallel processing at present is the large investment in time and money that is often required to change the operation of sequential code to that on a parallel computer eg an array of transputers. This usually means at best, making changes to the source code or at worst rewriting the code in another language eg Occam. In the academic world availability of the user source code is usually not a problem and given the necessary time and effort the code can be sufficiently changed to run on a taskfarm environment. In a commercial situation, however, with code often costing hundreds of thousands of dollars, or with commercial secrecy a problem concerning the operation of software, source code availability is not guaranteed. As a result it is essential that commercial code should be able to run on the taskfarm and all that would be required would be recompilation of existing code on the transputers, something that could easily be carried out by the company itself.

For reasons already discussed it was also considered important to be able to have a general taskfarm system that could be easily adapted to be used with any user software. This would again reduce the time and effort required in porting code onto parallel machines.

By providing the facility of a control file, any changes that are then required to the system can be implemented by a minor editing change of a single file.

Problems to Overcome

Given the strict guidelines outlined in the previous section, many problems presented themselves. The following section describes the major problems to be overcome.

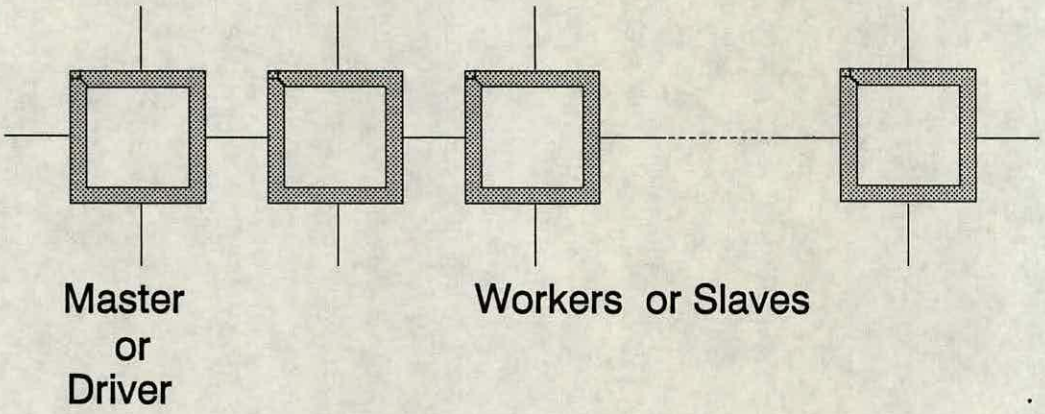


Figure 6–7: A general outline of the taskfarm structure

1. Due to an identical copy of the code running on each worker transputer, all workers would try to read from the keyboard, write to the screen and read/write from/to the same files. It was essential that any solution would remove such a scenario.

2. Since each worker thinks that it is the only code running and that it alone has complete access to the filing system, all workers must be given a pseudo access to the filing system. This will require some form of communication system to pass file requests to and from the filing system from each worker.

3. For large jobs ie with a large number of worker transputers, there may be a limit to the number of files that can be kept open at any one time. For example if the maximum limit of the system in question is 32 files and there are 16 workers this means that only two files can be open at any one time on each worker. For a PC system the maximum value that can be opened is only 16. It is important that this limitation should be allowed for in the taskfarm.

The taskfarm will have the general form of a driver (or master) transputer allocating tasks to a number of worker (or slave) transputers. This is outlined in figure 6–7.

Since each worker needs access to the host filing system, an efficient message passing system will be employed to send and receive messages from the worker to the driver and back. It was decided to use TINY [77] as the communications package and 3L Parallel C [88] to control the farming process. The harness was originally written for a PC transputer system using the 3L Toolset utilities and in particular, the *afserver*.

Before giving a detailed description of the taskfarm and how the problems were overcome, it is necessary to describe more fully the working of TINY and the afserver.

6.3.2 TINY

TINY was developed at the Edinburgh Parallel Computing Centre and has the great attraction of being highly portable across a number of different hardware platforms and can be used to send messages efficiently on an arbitrary topology. TINY works on the principle of each process having a unique ID. To send or receive messages across the network all that is required for the user is the process ID, the name of the buffer containing the information to be sent, the size of the buffer and the type of message to send. Because TINY can be used on any topology it firstly explores the network to establish the routing tables. Thus in summary TINY can be used to:

- Determine the processor topology
- Determine the process mapping
- Calculate the routing tables
- Start the message router process
- Read and write messages

The first three tasks are performed by calling an initialise procedure, the router process is started using a separate call and the user is responsible for calling procedures for the reading and writing of messages.

There are three routing strategies implemented by TINY.

Adaptive The message is sent to a particular destination process and decisions are taken at each node as to the route to be taken. The order in which the messages arrive at the destination cannot be guaranteed using this strategy.

Sequential This is the same as adaptive routing except the message follows a pre-defined route, thus allowing control over the arrival of the message at the destination.

Broadcast As the name suggests this strategy permits the user to send a message to all other user processes on the network.

The user can call the required TINY procedures from C in order to send and receive messages. For the purposes of this work, sequential messages were passed around the network. To pass messages using TINY a specific protocol must be adhered to. The C code required to send and then receive a sequential message is outlined in figure 6–8.

`type` is the type of message to be sent or received, `dest` is the destination process identifier, `source` is where the message is coming from, `data` is a buffer for storage of the message to be sent or received and `length` is the length of the buffer.

Thus when each worker wishes to access the host filing system, all commands to be sent to the driver must be sent (and received) using the above protocol.

6.3.3 Afserver

The user simulation code to be implemented on the taskfarm is SSUPREM4 [16]. SSUPREM4, which is written in C, was compiled on the transputers using 3L

```
#include "tiny.h"

int type, dest, length;
int *source;
void *data;

int ok;

ok = t_sseq(type, dest, data, length);   /* Send message */

ok = t_recv(type, source, data, length);  /* Receive message */
```

Figure 6–8: Example C code for sending and receiving of sequential messages

Parallel C . Using the 3L utilities, namely the afserver (which is part of the Parallel C package), SSUPREM4 will communicate with the filing system at a low level, using a fileserver protocol. The function of the afserver is to provide an interface between the host filing system and the root transputer. This is outlined in figure 6–9. This shows how the filer process (something which allows the user to perform file operations) interfaces with the root transputer process. The communications takes place via two channels, **to.filer** and **from.filer**. The former sends commands from the root to the host filer and the latter from the host filer to the root.

The host filing system expects to receive information in the following file protocol format [101]. For example a 32 bit integer is sent as `int32.value;INT32`, where `int32.value` indicates that a 32 bit integer is about to be sent and `INT32` contains the actual value. A record is sent as `record32.value;INT32::[]BYTE`, meaning a record is about to be sent, followed by the size of the record, followed by the record itself. As defined by the protocol a **record32.value** has a byte value of 12. Provision is also made to send a zero-length record, **nilrecord**, which has a byte value of 8.

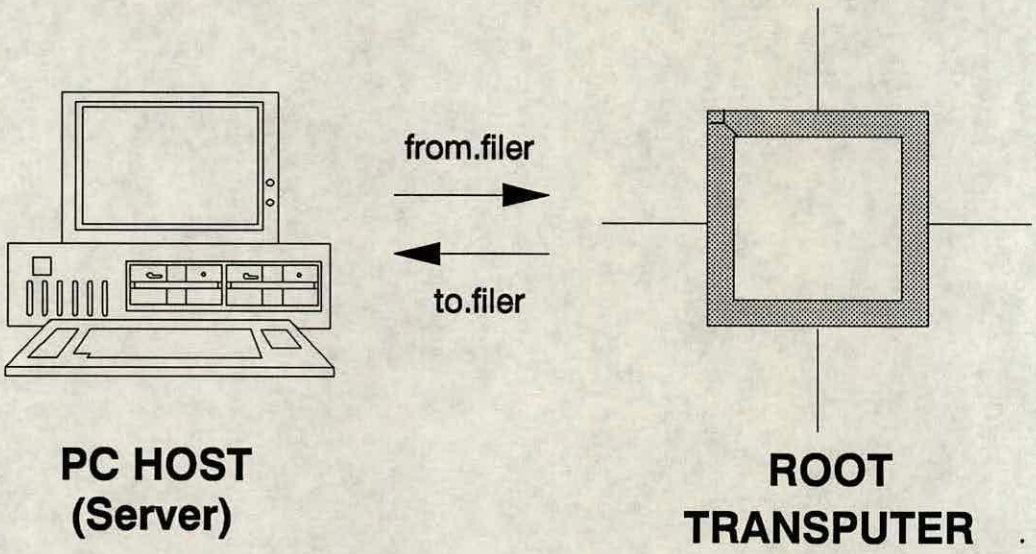


Figure 6–9: The general operation of the afserver

Thus in summary:

integer is defined as `int 32.value; INT32`

record is defined as `nilrecord.value` or
`record32.value;INT32::[]BYTE`

All the afserver commands or AFS commands (from the user process) will be made up of a combination of integers and/or records. The host server will in turn send the required combination of integers/records back to the user process. Examples of these commands include **OpenFile.Cmd, OpenInputStream.Cmd, ReadBlock.Cmd, WriteBlock.Cmd,** etc. The complete list of commands is detailed in [101]. Figure 6–10 illustrates in more detail the parameters required for the execution of the **OpenFile.Cmd.**

Thus each user process will communicate with the host filing system using these AFS commands and will send the **to.filer** command protocol and receive the **from.filer** protocol.

```
OpenFile.Cmd:

  to.filer.protocol = openfile.cmd (integer) filename (record)
                      access.method (integer) open.mode (integer)
                      exist.mode (integer) record.length (integer)

  from.filer.protocol = stream.id (integer) result (integer)
```

Figure 6–10: The parameters used in the OpenFile.Cmd

6.3.4 General taskfarm operation

This section seeks to address the problems detailed in section 6.3.1.

The general operation of the taskfarm can be considered as the host filing system talking with the root transputer, which in turn communicates with the various worker transputers. Each worker transputer will issue different AFS commands to the host filing system. In order for these commands to reach the host they must be sent using the TINY message passing system to the root transputer, since this is the only transputer that has direct access to the filing system. This results in a problem as the AFS commands have a particular protocol, as outlined in section 6.3.3, and TINY has its own protocol (see section 6.3.2), thus there must be a protocol conversion before messages can be sent and received around the network. This general structure is outlined in figure 6–11.

Each slave (or worker) in this configuration has a pseudo access to the host filing system, via the driver (or root) transputer.

Operation of the worker process

As the name suggests, the worker process is resident on all the worker transputers on the taskfarm. A detailed illustration of the operation of the worker process is outlined in figure 6–12.

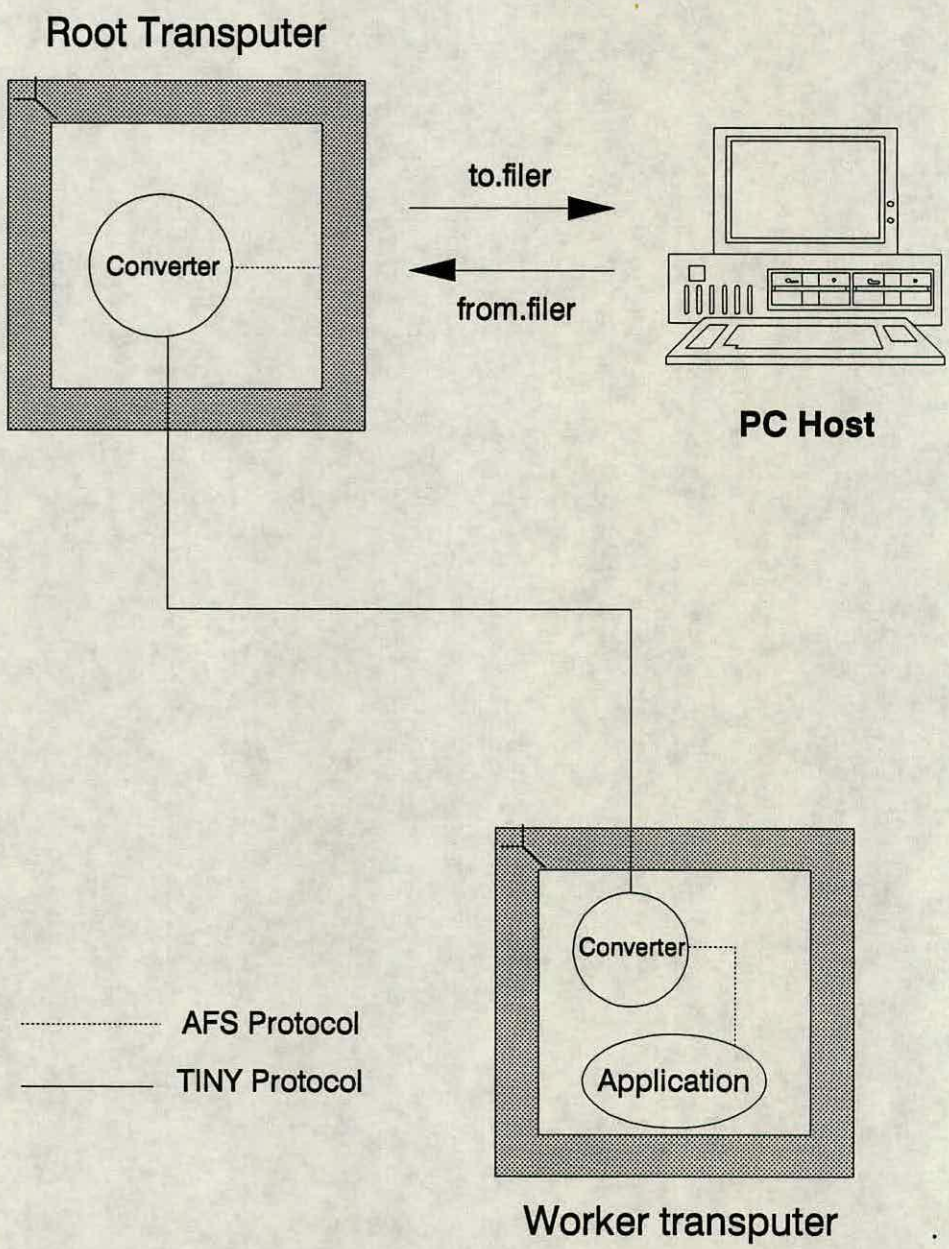


Figure 6–11: The general structure of the taskfarm showing the AFS protocol and the TINY protocol

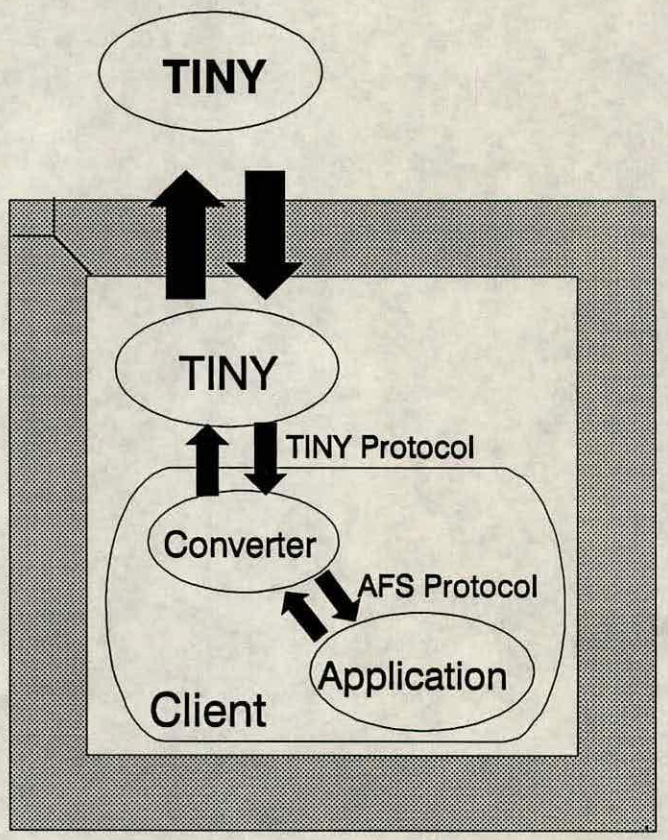


Figure 6–12: A detailed illustration of the worker process

The application program, SSUPREM4, communicates with the client program via a pair of communication channels, using the AFS protocol and is unaware that it is not directly connected to the filing system. Since the worker process cannot use standard C I/O functions, eg printf, it is described as a *standalone* task. This means that it is linked with a standalone version of the runtime library, which does not support standard I/O functions.

The general flow of control is as follows:

- Application executes AFS command.
- Client program "captures" AFS command.
- Client makes any necessary changes to the command.
- Client converts from AFS protocol to TINY protocol.

- Client sends command to driver.

- Client receives acknowledge from driver in TINY protocol.

- Client converts TINY protocol acknowledge to AFS protocol and sends to application.

Since this process is taking place on all the worker transputers, great care needs to be taken to ensure that the correct workers are opening and closing the correct files. This problem is increased when we consider the fact that all workers will be trying to read from the same keyboard, write to the same screen and open the same system files for eg data output. Given the initial requirement that no access would be given to the source code, the file system problem had to be overcome using the client and driver process.

Section 6.3.5 outlines in more detail the control within the taskfarm and how the user can specify filename alterations. Suffice it to say at this stage, each client process will have a table of all the necessary filename translations required for the particular application. Thus for example if worker number 1 requests to open the file ssup4ukey, the client process will look up its file table and find the equivalent filename translation eg ssup4ukey1 and proceed to open the file ssup4ukey1 as normal. This process is repeated among all the workers, ie worker number 2 opens file ssup4ukey2 etc. As a result each file that is opened is given a unique name and a unique stream id number.

The procedure is even more complex when the client receives an **OpenInputStream.Cmd**, with stream 0. This is a request for the application program to read from the keyboard. Since the standard input is to come from a file, it is necessary to first look up from the file translation table the name of the standard input file and then perform an **OpenFile.Cmd** instead of the **OpenInputStream.Cmd**. It is essential, however, to send back to the application code an acknowledge for the original **OpenInputStream.Cmd** and not the **OpenFile.Cmd**.

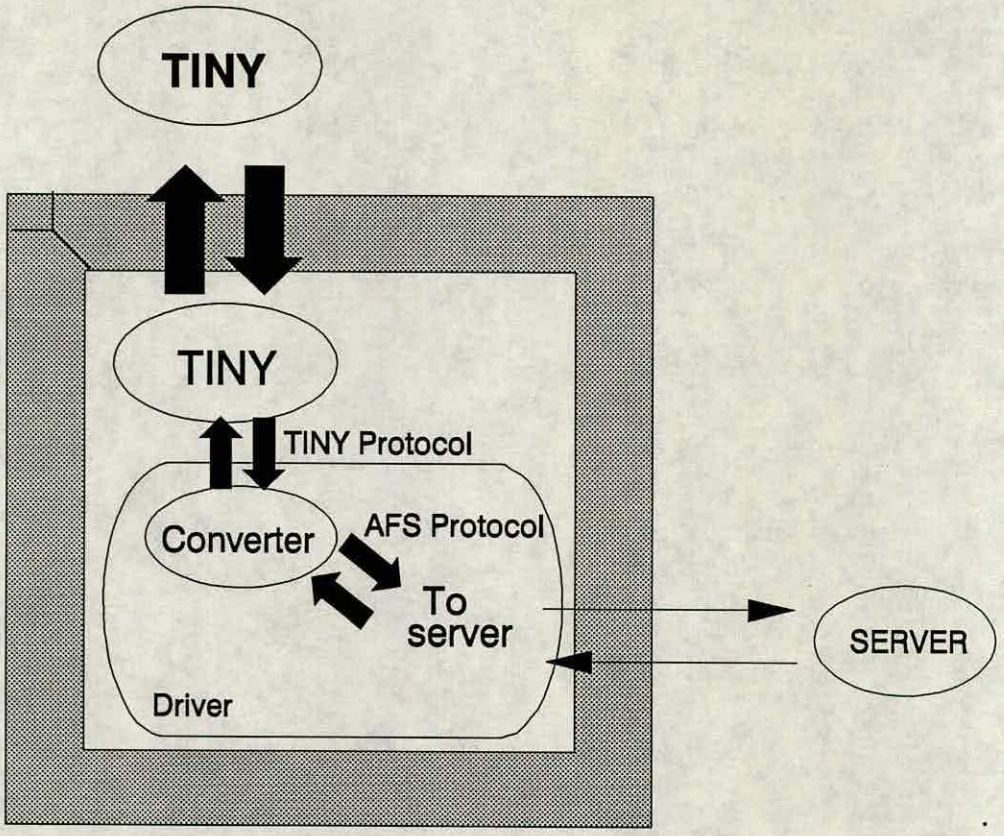


Figure 6–13: Detailed operation of the driver process

It is the low level control offered by the altering of the AFS protocol in the manner described above that permits the operation of the taskfarm without modifications to the source code.

Operation of the driver process

The driver process is responsible for controlling the whole taskfarm and is resident on the root transputer. The operation of the driver process is outlined in figure 6–13.

The driver process receives the TINY protocol AFS commands from the different workers. Since most of the necessary alterations have been made to the AFS commands at the worker, the driver will, in most cases, convert the TINY protocol to an AFS protocol and execute the required function on the filing system. The **OpenFile.Cmd** is not so straightforward, however. Due to the limitations on the

maximum number of files that can be kept open at any one time, not all openfile commands can proceed as normal to the server, otherwise the system limit will be reached very quickly, resulting in a system crash.

The maximum number of files that can be open simultaneously using DOS is 16, so if using a PC system, some files will have to be closed before others can be opened. The logic to control this is included in the driver process. When the driver receives an **OpenFile.Cmd** it firstly checks to see if the maximum file limit has been reached, if not then the file is opened as normal. If, however, the limit has been reached then the driver will initially close a low priority file, (one which is not used frequently) storing its present file pointer position. This will then allow the new file to be opened and an acknowledge sent back to the worker. As far as the worker is concerned, it is unaware of this and thinks that all files are still open.

Although this opening and closing of files, as well as the TINY to AFS protocol conversion, exhibits a finite overhead on the system, section 6.3.6 shows that for CPU intensive applications, this becomes less significant when contrasted with the time spent on computation as opposed to "communication".

Another function of the driver process is to dynamically allocate the simulation code to the worker transputers. Initial results showed that the speed of loading large simulation packages from disk was causing a bottleneck. The time taken to load SSUPREM4 took about 150 seconds per worker, so for a 10 worker farm, 25 minutes would be spent loading the executable code from disk. Extra logic was added to the driver which firstly loaded the code into the memory of the driver, then, assuming each worker requires this same code, copied the program from the driver's memory to that of the worker. This reduced the time taken to load the code from 150 seconds to about 5 seconds per worker. Thus the total time spent loading for 10 workers is now $150 + (9 \times 5)$ ie 195 seconds, quite a significant reduction. Only when new code is required will the original disk load be necessary.

6.3.5 System control

In order to simplify the ease of use of the system, user control was implemented via a single control file. Thus any changes required by the user eg change of application code can be brought about by a minor editing change to this file.

An example of part of a control file used for SSUPREM4 is shown in figure 6–14.

The **BLOCK** statement allows control of the jobs to be submitted and block 2 will not commence until block 1 is finished. This is particularly useful for simulations, when the device simulation programs cannot begin until the process simulations have been completed. The **do** statement specifies the program to run on the farm, **assign** renames any necessary filenames and **stdin** and **stdout** allow for the redirection of the standard input and output to the specified filenames. As a result the taskfarm is not restricted solely to simulation packages but can be used for any user code.

6.3.6 System Results

The original taskfarm was written for an IBM PC with a rack of transputers containing five T414 transputers, each with 2 Mbytes of memory, and eight T800's, each with 256 Kbytes of memory. The original code used with the taskfarm was SSUPREM4 (this source code was now made available). The copy of the source code to be compiled for the transputers was a version written in C for execution on a SUN4 workstation. Major porting problems, however, restricted the ease and efficiency of the recompilation process. The original copy of the software contained many UNIX system calls, something which is not supported in the 3L implementation of C for the PC. As a result many system routines had to be rewritten to allow execution on the transputer system. Differences in the two compilers also resulted in unpredictable behaviour of the executable code. Although the recompilation process can in theory be dismissed as trivial, in practice this is often a non-trivial task, especially if the code to be ported is system specific and the inherent differences between the source and target architectures are great. On the

```
begin BLOCK-1
do s4.b4
assign ssup4uskey ssup4uskey1
assign complete.dat complete.dat1
stdin run01.in
stdout run01.out
done
do s4.b4
assign ssup4uskey ssup4uskey2
assign complete.dat complete.dat2
stdin run02.in
stdout run02.out
done
do s4.b4
assign ssup4uskey ssup4uskey3
assign complete.dat complete.dat3
stdin run03.in
stdout run03.out
done
end
begin BLOCK-2
do s4.b4
assign ssup4uskey ssup4uskey1
assign complete.dat complete.dat1
stdin run01.in
stdout run01.out
done
end
```

Figure 6–14: An example of a control file for SSUPREM4

other hand if code is written only to adopted standards and compilers adhere to these standards, then porting problems will be kept to a minimum or disappear altogether.

It should be noted here that the only changes made to the source code were to overcome system dependencies and no modifications were made to the filenaming nomenclature.

One of the drawbacks of utilising this form of taskfarm is that each transputer must have a large local memory, since it is to execute a complete copy of the program. When using large simulation packages, the need for large amounts of local memory becomes more acute, since these are "memory hungry" packages. This problem was immediately evident when attempting to run SSUPREM4 on either the T414's or the T800's on the PC. In the case of the T800's (which only have 256 Kbytes of RAM) there was not even enough memory to store the executable code. Although the T414's could load the executable code, there was not enough memory to allow allocation of all static arrays, let alone dynamic memory requirements at runtime.

In order to prove the adequacies of the taskfarm with the SSUPREM4 code, two options were available:

1. Obtain more transputers with much more memory for the PC system.

2. Change to Edinburgh Meiko Computing Surface which has 17 T800 transputers each with 8 MBytes of memory.

Since the latter did not involve any extra cost, this route was chosen. It was, however, necessary to port the taskfarm from the PC system to the Meiko Computing Surface. As a result all PC system dependencies had to be rewritten or excluded from the taskfarm to facilitate its operation on the Meiko.

The general structure of the Meiko Computing Surface is outlined in figure 6–15.

The domain structure allows users to log onto a particular domain to develop or run applications. One of the domains contains 17 T800's each with 8 MBytes

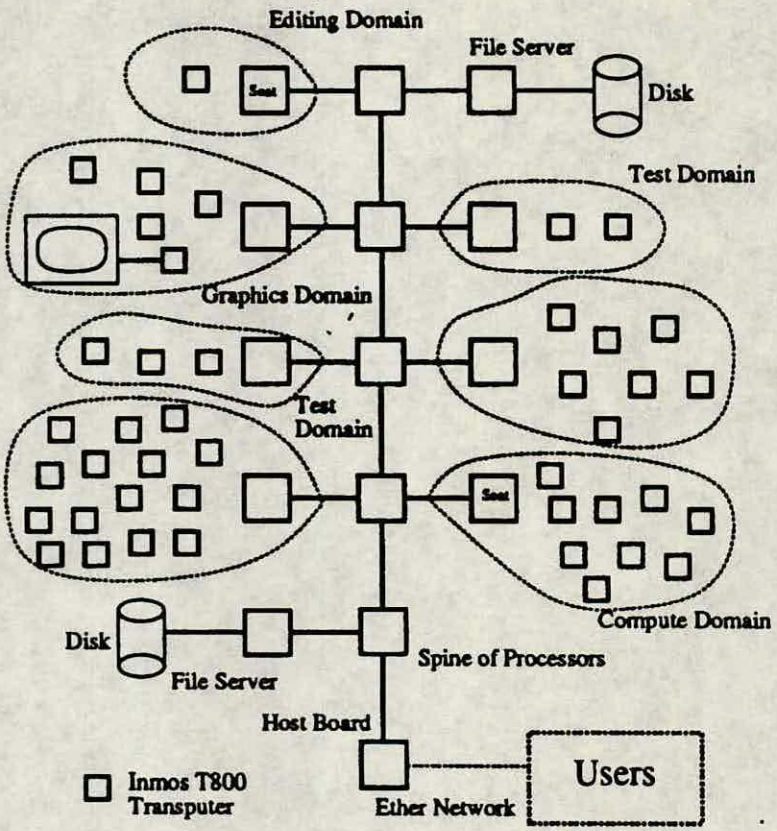


Figure 6–15: The general structure of the Meiko Computing Surface at Edinburgh

of memory and it was this domain that was used in the operation of the taskfarm. The general structure of the taskfarm itself is unchanged to that outlined in figure 6–9, except the host is no longer the PC but a transputer, connected to the Meiko filing system.

The larger memory on the Meiko Computing Surface thus enabled SSUPREM4 to be executed using the taskfarm.

6.3.7 Timing results

Two different topological configurations of transputers were initially investigated on the Meiko, namely a pipeline and a tree configuration. These configurations were detailed in figure 5–6. Since the topology of the transputers on the Meiko are software configurable, this lends itself easily to investigation or alteration of

```
5
0 1 1 0
1 1 2 0
2 1 3 0
3 1 4 0
4 1 5 0
```

Figure 6–16: An example of a wire file used on the Meiko Computing Surface

	Tree/min	Pipeline/min	%Speedup
Short	13.62	14.17	4.0
Long	721.7	722.7	0.14

Table 6–1: A comparison of a tree and a pipeline configuration for two different simulation files

hardware configurations. The topology can be readily changed using *wire* files. An example of such a file is outlined in figure 6–16.

This means that link 1 of transputer 0 is connected to link 0 of transputer 1, etc. Table 6–1 shows the results obtained for running 10 short simulations and 10 longer simulations on 10 transputers, using a tree and a pipeline configuration.

As can be seen from the table, for short simulations there is a modest speedup of 4.0%, for the longer simulation, however, the speedup is negligible. This is due to the fact that the communications bottleneck, eased by the tree structure, is less significant in the case of the longer simulation, due to the greater length of time performing computation.

Detailed results for the short simulation using the tree and pipeline configurations are outlined in figure 6–17 and figure 6–18 respectively.

Careful examination of figures 6–17 and 6–18 show that the time taken for loading the code on the tree is less than that of the pipeline. This is because, for the pipeline, the code for the workers towards the end of the chain must be routed

```
Edinburgh Microfabrication Facility
Simulation Taskfarm System
System Nodes : 10
Batch File   : sp4_10.rc

Node  Program   Calc Comm Load Misc
==== ========= ==== ==== ==== ====
  3   s4.b4     30  589  150    0
  1   s4.b4     30  542    0    0
  7   s4.b4     30  354    1    0
  8   s4.b4     30  259    1    0
  2   s4.b4     30  496    0    0
  6   s4.b4     30  212    1    0
  4   s4.b4     30  401    1    0
 10   s4.b4     30  449    1    0
  5   s4.b4     30  307    1    0
  9   s4.b4     30  165    1    0

Completed 10 jobs : Elapsed time 817 seconds
```

Figure 6–17: Detailed results for a short simulation, using a tree configuration

through the preceding workers. Thus the time taken to load worker 10 will be greater than the time taken to load worker 2.

To summarise, when even longer simulations are being carried out, the difference between the two topologies will be even more negligible. As a result for the remainder of this section, all results will be detailed for the pipeline configuration.

The following results were obtained using the Meiko Computing Surface and SSUPREM4. It should be noted that in the following section all times for a SUN4 are actual CPU times. Real user time or elapsed time is much greater.

Figure 6–19 illustrates the results obtained by taking different sizes of simula-

```
Edinburgh Microfabrication Facility
Simulation Taskfarm System
System Nodes : 10
Batch File   : sp4_10.rc

Node  Program   Calc Comm Load Misc
==== =========  ==== ==== ==== ====
  2     s4.b4    30  572    1    0
  4     s4.b4    30  475    3    0
  5     s4.b4    30  426    3    0
  3     s4.b4    30  524    2    0
  1     s4.b4    30  620  152    0
  6     s4.b4    30  376    4    0
  7     s4.b4    30  325    5    0
  8     s4.b4    30  273    6    0
  9     s4.b4    30  221    7    0
 10     s4.b4    30  167    7    0

Completed 10 jobs : Elapsed time 850 seconds
```

Figure 6–18: Detailed results for a short simulation, using a pipeline configuration

tions and executing them on a 10 transputer taskfarm. For comparative purposes the time taken to execute 10 simulations sequentially on a SUN4 is also included.

As the task number increases, so does the length of the simulation. For the longer simulations the communications overhead becomes increasingly less significant compared to the computation, hence the savings to be made with the taskfarm become more evident. The benefit of the taskfarm is more apparent when one considers that for these types of simulations a single SUN4 is about three times faster than a single T800.

Figure 6–20 details the results obtained by simulating 10 jobs on varying num-

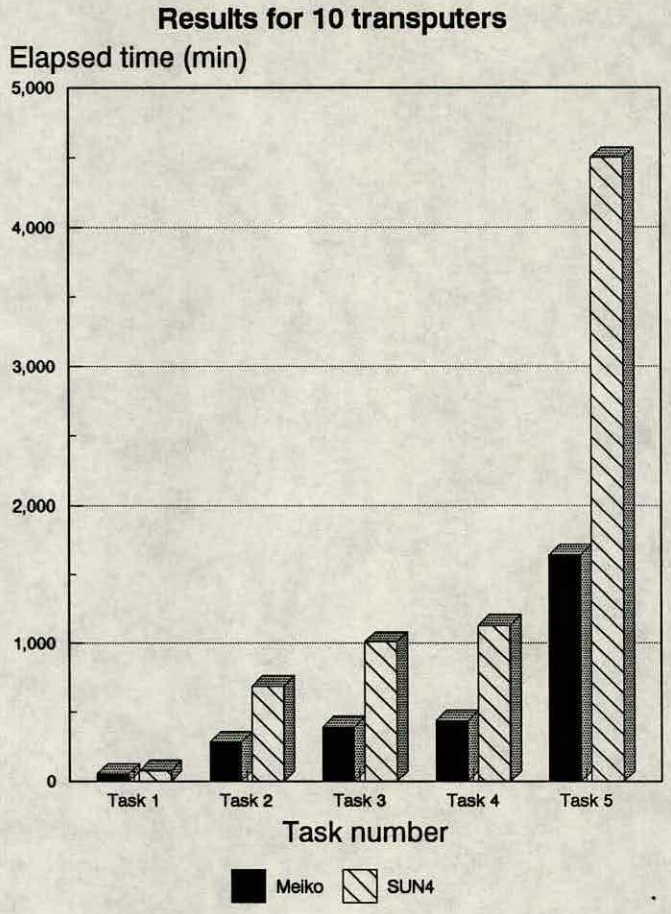


Figure 6–19: Results obtained for a 10 transputer taskfarm and the time taken to run 10 jobs on a SUN4 workstation. As the task number increases so does its computational requirements

bers of transputers. Again a reference is included to show the time taken to run 10 jobs sequentially on a SUN4 workstation.

For example, in order to run 10 jobs on only 2 transputers, this will require 5 iterations, whereas for 10 transputers only one iteration is required. Following this logic, the times taken for 6 and 8 transputers should be the same, as both require two iterations. The results for these, however, are quite interesting and are detailed in figure 6–21 and 6–22

The results for the 8 transputer farm show a slight improvement in elapsed time, compared to the 6 transputer farm. This is because for the 8 transputer example only 2 transputers will be communicating with the driver in the second

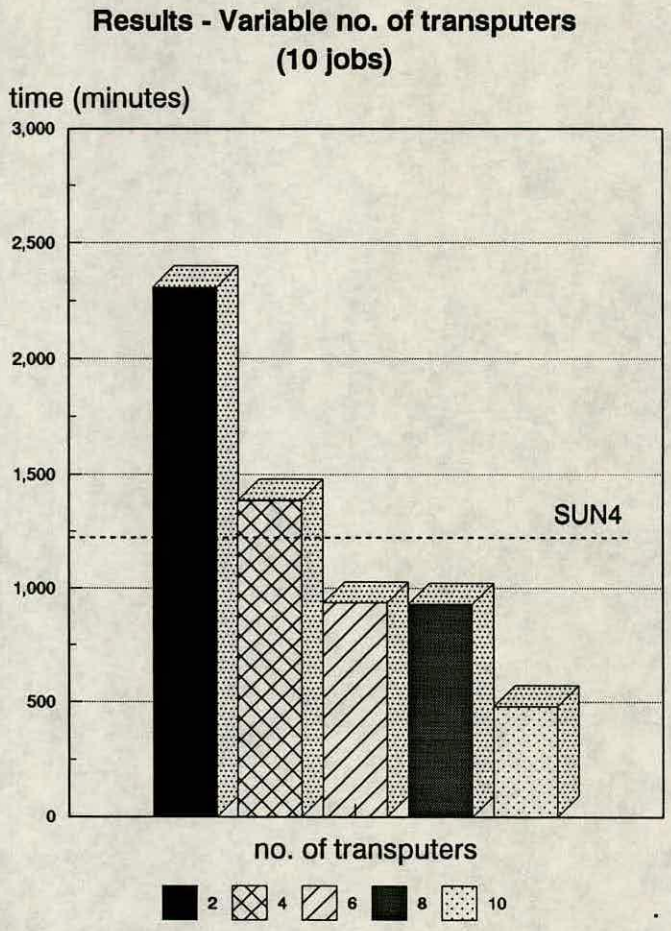


Figure 6–20: Time taken to run 10 jobs on variable numbers of transputers

iteration, hence decreasing the communication overhead. This can be seen by comparing the "comm" times for both figures 6–21 and 6–22. In the case of the latter only 46 seconds is taken up on communications as opposed to about 200 seconds for the former.

The next stage in the procedure was to take the device simulation package, PISCES-2B [17], and compile this on the transputers. This was completed but, due to the large memory requirements of the program, it was not possible to use this on the taskfarm. This problem was further exacerbated as PISCES (which is written in FORTRAN) has large numbers of **integer*2** arrays. Since the 3L Fortran compiler does not support 2 byte integers, all integer arrays were effectively doubled to accommodate the 4 byte integer syntax.

Section 6.4 describes how PISCES-2B was used in a parallel taskfarm, as on

```
Edinburgh Microfabrication Facility
Simulation Taskfarm System
System Nodes : 6
Batch File   : sp4_10.rc

Node  Program   Calc   Comm   Load   Misc
==== ========= ====   ====   ====   ====
  2     s4.b4   27617   431    1     0
  3     s4.b4   27617   415    2     0
  1     s4.b4   27617   514  149     0
  4     s4.b4   27617   380    3     0
  5     s4.b4   27617   346    3     0
  6     s4.b4   27617   296    4     0
  2     s4.b4   27617   129    0     0
  3     s4.b4   27617   212    0     0
  1     s4.b4   27617   209    0     0
  4     s4.b4   27617   201    0     0

Completed 10 jobs : Elapsed time 56157 seconds
```

Figure 6–21: Detailed results obtained for 6 transputers running 10 jobs

this parallel machine each node possessed 16 MBytes of memory and the compiler supported 2 byte integers.

In conclusion, the taskfarm described offers a very simple and efficient method of parallel processing. Given a PC and a number of transputers, which can be plugged into a slot on the PC, the user has access to a very cheap desktop supercomputer. Using the taskfarm the field of parallel computing can be easily accessible to the user with computationally intensive applications. It is hoped that the need and potential of such a system has been adequately described in

```
Edinburgh Microfabrication Facility
Simulation Taskfarm System
System Nodes : 8
Batch File   : sp4_10.rc

Node  Program   Calc   Comm   Load   Misc
====  ========  ====   ====   ====   ====
  4     s4.b4   27617   294     3     0
  1     s4.b4   27618   440   149     0
  2     s4.b4   27617   475     1     0
  3     s4.b4   27617   437     2     0
  7     s4.b4   27617   240     5     0
  5     s4.b4   27617   342     3     0
  6     s4.b4   27617   292     4     0
  8     s4.b4   27617   189     6     0
  4     s4.b4   27617    46     0     0
  1     s4.b4   27616    46     0     0

Completed 10 jobs : Elapsed time 55918 seconds
```

Figure 6–22: Detailed results obtained for 8 transputers running 10 jobs

this section and given extra memory, this system could easily be used for serious semiconductor optimisation and simulation.

6.4 A parallel taskfarm using Intel i860's

This section seeks to describe a similar taskfarm to that described previously, but with three major differences.

1. Each worker processor is an Intel i860 as opposed to a transputer.

2. Changes can be made to the source code for filenaming purposes.

3. CS Tools will be used to control the taskfarm.

The machine on which the simulations are to be performed is at the Edinburgh Parallel Computing Centre and is called *Maxwell.* It is a Meiko Computing Surface with 64 Intel i860 processors, each containing 16 MBytes of memory. It has a SPARC front end running SUN OS, with 4 Gigabytes of disk space and a peak performance of 5 Gigaflops.

The taskfarm will operate in a similar manner to that described in section 6.3.1. Each worker processor will execute a sequential copy of the simulation program. As a result logic will need to be added to ensure that not all the workers read from/write to the same files. This can be provided by utilising CS Tools and will be outlined in detail in the following section.

6.4.1 General operation of the taskfarm

The code used in the taskfarm was SSUPREM4 and PISCES-2B. Initially S-SUPREM4 was taken and compiled on Maxwell. As was the case for the transputer taskfarm, system dependencies had to be rewritten or commented out of the code to allow its use on Maxwell. The Maxwell C compiler, however, supports a larger variety of system calls, compared to the 3L C compiler, thus assisting in the porting of the code.

Once a sequential copy of SSUPREM4 was running on a single i860 processor, it was essential to then add the necessary logic to permit multiple copies to run on a number of worker processors.

Before a particular worker processor opens a specific file, it is necessary to give that file a unique name. This is achieved using CS Tools. Using the **cs_import** command [102], the user can determine the processor number, and since each simulation is performed on separate worker processors, this number will always be unique.

```
#include <sys/types.h>
#include <cstools/cstools.h>

main ()
{
.......
int lprocid; /*Defines integer*/
char key_id[12],mod_id[12]; /*Defines characters*/
char imp_id[12],outfile[12]
.....
cs_import("int procId",&lprocid); /*Get worker id number*/
sprintf(key_id,"ssup4uskey%d",lprocid); /*Tag on worker id*/
sprintf(mod_id,"ssup4mod%d",lprocid);
sprintf(imp_id,"ssup4imp%d",lprocid);
sprintf(outfile,"ssup4out%d",lprocid);

freopen(outfile,"w",stdout); /*Redirect stdout to outfile*/
.....
}
```

Figure 6–23: Part of the C program required to implement the filenaming system

For example, if worker number 5 wished to open the file ssup4ukey, then it would call **cs_import** to establish the processor number, tag this onto the end of the file in question and proceed as normal. In this case the worker would request to open the file ssup4ukey5. Figure 6–23 outlines part of the C program required to implement this logic.

Essentially the program:

- Defines the necessary parameters.
- Gets the worker unique id number.

```
par
    processor 1 for 4 (proc_type i860) ssuprem4 test.dat
endpar
```

Figure 6–24: An example of a PAR file

- Tags the worker id number onto the end of the four files to be opened.

- Redirects the stdout (screen) to a specific output file.

Thus provision is made for all file renaming, except the required input file. This provision is outlined below.

6.4.2 Running the taskfarm

Before detailing how the programs are executed on the farm it is necessary to discuss the concept of a *par* file [103]. PAR files are part of the CS Tools programming utilities and are a representation of the structure of parallel programs. An example of a PAR file is outlined in figure 6–24.

This allocates four different worker processors and each processor runs S-SUPREM4 on the file **test.dat**

In order to execute the PAR file, it must be passed to another CS Tools utility called *mrun*. This is a parallel loader which is used to load either executable files or PAR files onto the Computing Surface. If the argument passed to mrun is not executable, then it assumes that it is a PAR file. Thus, to farm the simulations across the network, the following simple example command needs to be initiated:

mrun ssuprem4.par

This will execute the information detailed in ssuprem4.par given the number of specified processors.

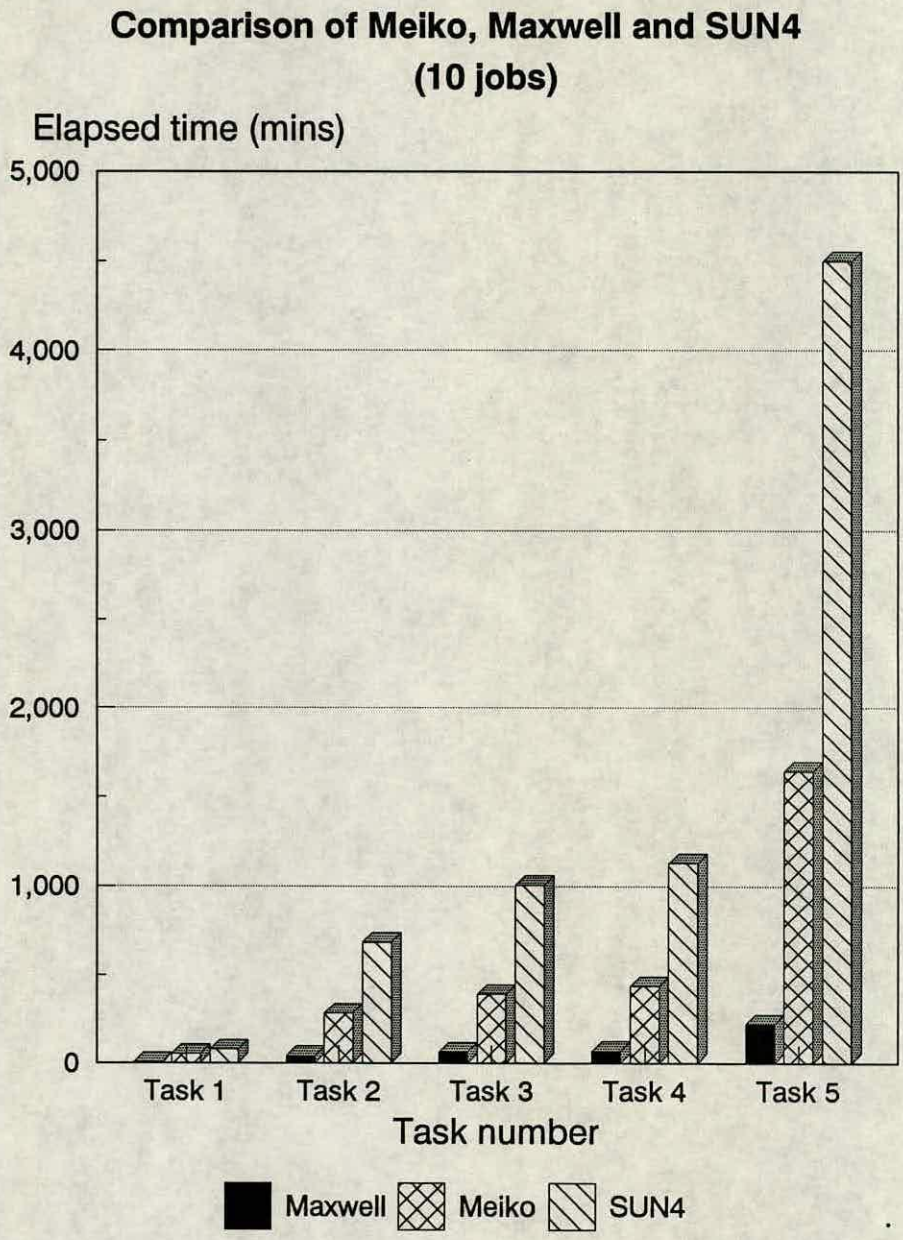


Figure 6–25: Results for a 10 processor taskfarm on Maxwell, 10 processor taskfarm on the Meiko and 10 sequential simulations on a SUN4

Figure 6–25 shows the results obtained, after running SSUPREM4 simulations on a 10 processor taskfarm. For reference purposes the times taken to run the same files on the transputer taskfarm and on a SUN4 are also outlined. As before, the time shown for a SUN4 is the time taken to run 10 sequential simulations one after the other.

The next stage in the procedure was to compile PISCES-2B (which is written in

Fortran). A similar approach was adopted for this as for SSUPREM4. All system dependencies were removed and **cs_import** was used to obtain the worker id. This enabled each worker to translate the necessary common filenames to unique identifiers. This process was complicated further in the case of PISCES, due to the large number of temporary files that are required to be opened at runtime. On completion of this the taskfarm was executed as before, using mrun and PAR files.

Figure 6–26 shows the results obtained when 10 SSUPREM4 simulations were farmed out and the doping profiles were then loaded into PISCES for threshold voltage analysis and 10 device simulations were then executed on the farm. For comparison purposes, also detailed is the time taken to execute these 10 process and 10 device simulations sequentially on a SUN4 and also on a SUN4 farm. The latter will be outlined in more detail in section 6.5.

As was the case for the transputer taskfarm, Maxwell, or smaller versions of it, offer a relatively inexpensive entry into supercomputing performance. The following section briefly outlines a much simpler, but much more expensive alternative to either of the two taskfarms discussed previously.

6.5 SUN workstation Networking

Due to the good computing facilities available at the Electrical Engineering Department at the University of Edinburgh, it was also possible to run simulations in parallel on a network of SUN4 workstations. Previously all the times detailed for a SUN4 have been the CPU times recorded to run 10 sequential jobs on a SUN4. In this section the times recorded for a SUN4 farm will denote the time taken for the longest job to finish when the 10 simulations have been run in parallel on 10 workstations. This is effectively the elapsed time and will be dependent on the loading of the machines in question.

The procedure to execute the simulations in parallel is very straightforward and is a replication of the command below.

rsh -n almer "cd suprem;ssuprem4 test.dat test.out" &

This utilises the remote shell command, executing the specified command on the particular machine. Thus to run 10 simulations on 10 machines this general command can be replicated 10 times, with different hostnames, for different machines.

Figure 6–26 shows the results obtained on the SUN4 network after 10 process and 10 device simulations were performed in parallel. For comparative purposes the results for the Maxwell taskfarm and the time taken to run sequentially on a SUN4 are also detailed.

Although the results for the SUN4 network are favourable when compared with the transputer taskfarm, the cost to obtain such performance is much higher in the case of the SUN4. Only 10 processors have been used in all these timing comparisons. The advantage of the taskfarm approach is the low cost of additional processors. An additional 10 transputers could cost about 5000 pounds whereas 10 further SUN4 workstations could cost in the region of 50 000 pounds.

The results for the Maxwell taskfarm show a considerable time savings when compared with the SUN4 farm. This is due to the increased processor power of the i860 and also the dedicated architecture of the computing surface.

6.6 A comparative study of hardware platforms

This brief section details the timings obtained when running a floating point intensive program across a number of different platforms. Since it was not possible to run SSUPREM4 or PISCES-2B on the PC transputer system (due to lack of memory) this allows direct comparison of different systems.

Figure 6–27 shows the elapsed times recorded in running 5 floating point intensive programs on the following platforms.

- PC transputer taskfarm.

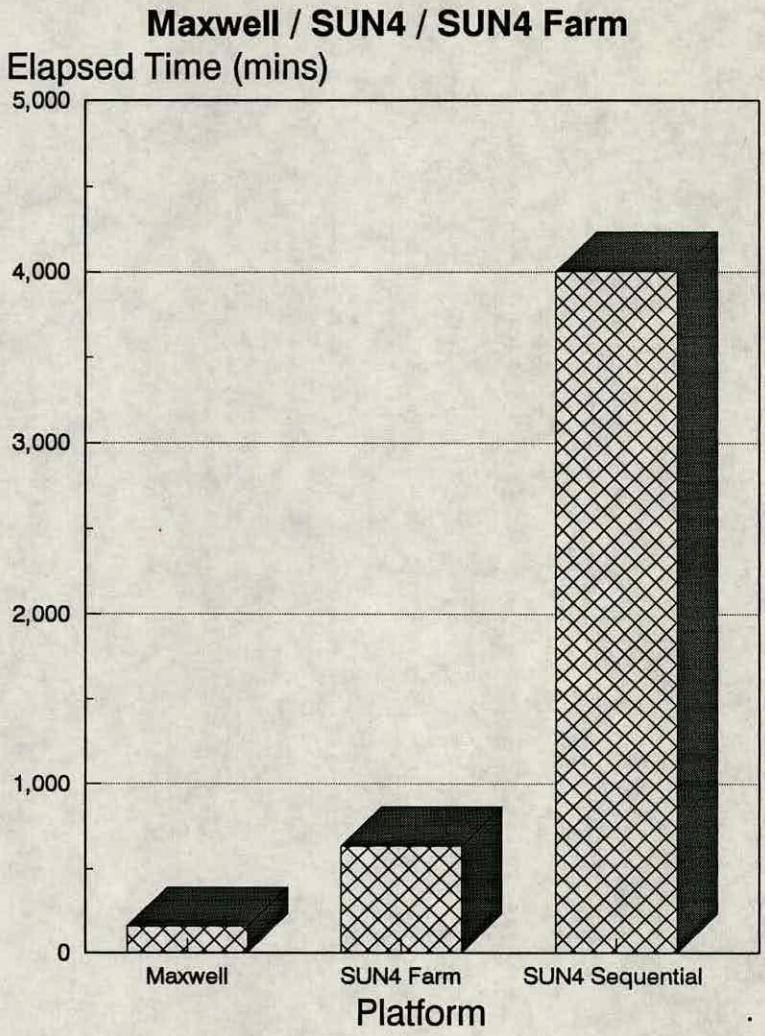


Figure 6–26: Results obtained for 10 parallel process simulations followed by 10 parallel device simulations on a network of SUN4's,on the Maxwell taskfarm and sequentially on a SUN4

- Meiko transputer taskfarm.
- Maxwell taskfarm.
- 5 sequential executions on a SUN3.
- 5 sequential executions on a SUN4.
- Parallel execution on a SUN4 network.

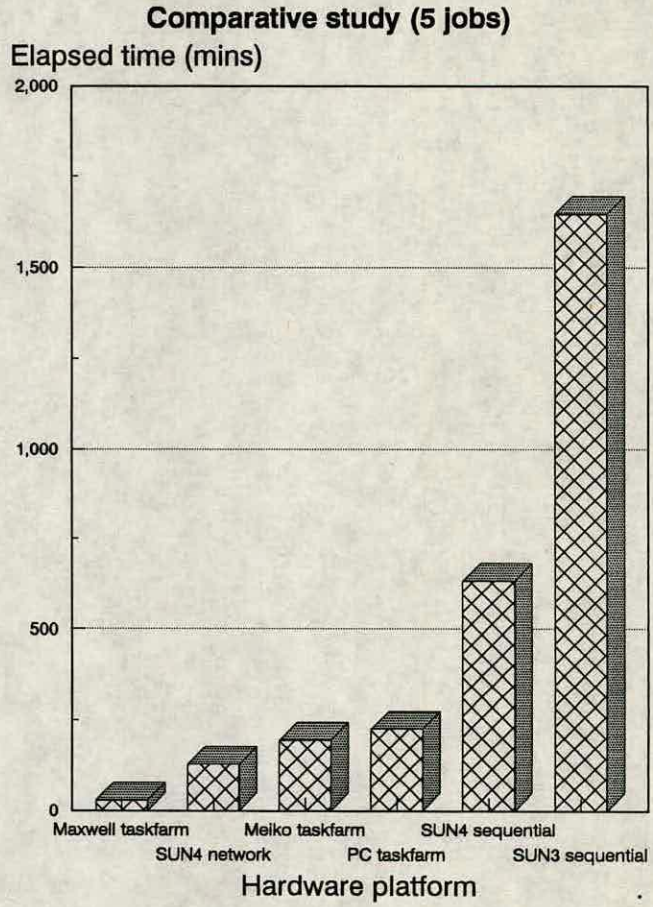


Figure 6–27: Results recorded for executing 5 programs on a variety of platforms

The Meiko taskfarm shows increased performance over the PC system due to the faster hardware on the Meiko compared to that on the PC.

6.7 Conclusions

Despite the great advances being made in processor speed and operation, there are still many computationally intensive tasks that require parallel or concurrent processing techniques to lead to realistic execution times. This chapter has sought to show how parallel computing can be used in the field of semiconductor simulation and decrease the time taken to obtain the final results. The most cost effective solution is to employ a transputer or Intel i860 taskfarm, however, given adequate resources a network of high performance workstations will lead to more than ade-

quate results. When large numbers of simulations are required to be performed eg 100, then a transputer/i860 taskfarm is by far the most efficient solution for this type of processing operation. Even for the transputer/i860 taskfarm approach the decision still has to be made between the cheaper, but slower, transputer or the faster, but more expensive, i860.

Chapter 7

Simulation and 1.5μm nMOS process optimisation

7.1 Introduction

Due to the ever increasing demand on the semiconductor industry, design engineers are being asked to produce new processes in record efficient time and to ensure that as the first wafers are produced, the quality and reliability is of a very high standard. As previously mentioned, simulation programs can be used to decrease the development time of the product, however, it is only when linked with statistical tools such as Response Surface Methodology (RSM) that truly efficient, reliable and robust designs can be formulated.

The traditional experimental approach is too slow and expensive to be used in today's competitive environment.

This chapter outlines how the simulation programs, SUPREM4, PISCES-2B and TOPEX can be used to generate information required by the RS/1 statistical package to optimise processes. The procedures will be illustrated for the Edinburgh Microfabrication Facility's 1.5μm nMOS process. Also described is the Taguchi method for optimisation and a comparison is made with the RSM approach. The major features of the process are outlined in chapter 8. An implant

step, which is used to control breakdown, was investigated along with its effects on specified device parameters. The parameters or responses under investigation were breakdown voltage, V_{bd}, gamma (the body effect coefficient), γ, subthreshold swing, S_t and threshold voltage, V_{th} [104]. The dose and energy of the implant were thus varied and the effects on the above parameters recorded.

SUPREM4 was used to model the effects of the different implant doses and energies on the doping profile, PISCES-2B was used to obtain values for breakdown voltage, subthreshold swing and IV curves. The latter were loaded into TOPEX to extract values for gamma.

Statistical tests were performed on the simulated data to establish the correctness of the fit and the suitability of the model.

The experimental objectives and response parameters will now be outlined in more detail in the following sections.

7.2 Experimental objectives and tradeoffs

The value of dose and energy for the double implant has a significant effect on the values of the aforementioned parameters. This section seeks to quantify the objectives of the experiment and outline the tradeoffs involved in the analysis.

The objectives can be summarised as follows:

- Ensure breakdown voltage is greater than or equal to 10 V.
- Minimise the value of gamma.
- Minimise the value of subthreshold swing.
- Ensure that the threshold voltage remains constant plus or minus 0.075V.
- Establish an optimum operating region.

- Perform sensitivity analysis to ensure the design is as insensitive as possible to fluctuations in the input parameters.

There are tradeoffs involved in the choice of implant dose and energy:

1. To ensure a large enough value of breakdown voltage in general requires specific values of dose and energy. These values depend on the range of dose and energy in question.

2. To minimise the value of gamma requires a low value of dose and energy, thus conflicting with 1. above.

3. To minimise the value of subthreshold swing requires a high value of energy.

4. Not all values of dose and energy will result in an approximately constant value of threshold voltage.

From the above analysis, the conflicts in the different design objectives or goals can be clearly seen. It is in the solution of such conflicts that RSM and its statistical tools can be effectively utilised. It should also be noted at this stage that the existing 1.5μm nMOS process has a second implant dose of 7×10^{11} atoms cm^{-2} and an energy of 140 keV.

7.3 1.5μm nMOS response parameters

The definition of the responses and the effects of implant dose and energy will now be described.

7.3.1 Breakdown

As device dimensions shrink and channel lengths are reduced the problem of breakdown becomes more apparent. Breakdown in short channel devices is generally considered to occur by two main mechanisms:

1. Punchthrough

2. Avalanche breakdown or source-drain breakdown

7.3.2 Punchthrough

Much has been written about the punchthrough mechanism [105-108] and the problems it causes in short channel devices. Punchthrough occurs when the depletion regions for the source and drain meet. The effect is caused by *drain induced barrier lowering* [109]. As the channel length decreases, more of the depletion region comes under the inversion layer. This results in an increase in surface potential, thus reducing the barrier to electrons. The difference in the depletion regions of a long and a short channel transistor are outlined in figure 7–1

As a result of the reduction in the barrier, electrons can now enter the region even at zero gate voltages. As the drain voltage is increased further the electrostatic potential will also increase, thus reducing the barrier even more, resulting in a greater punchthrough current. This process continues until the punchthrough condition is reached.

Punchthrough adjustment A simple equation relating punchthrough to various parameters is defined below [110].

$$V_{pt} = L_{eff}^2 N_a \frac{q}{2\epsilon_o \epsilon_{si}} \tag{7.1}$$

Where L_{eff} is the effective channel length, N_a is the substrate doping, ϵ_o is the permittivity of free space and ϵ_{si} is the relative permittivity of silicon.

From this and other analysis [109], the effects of punchthrough can be reduced by the following conditions:

- An increase in substrate doping

- An increase in the channel length

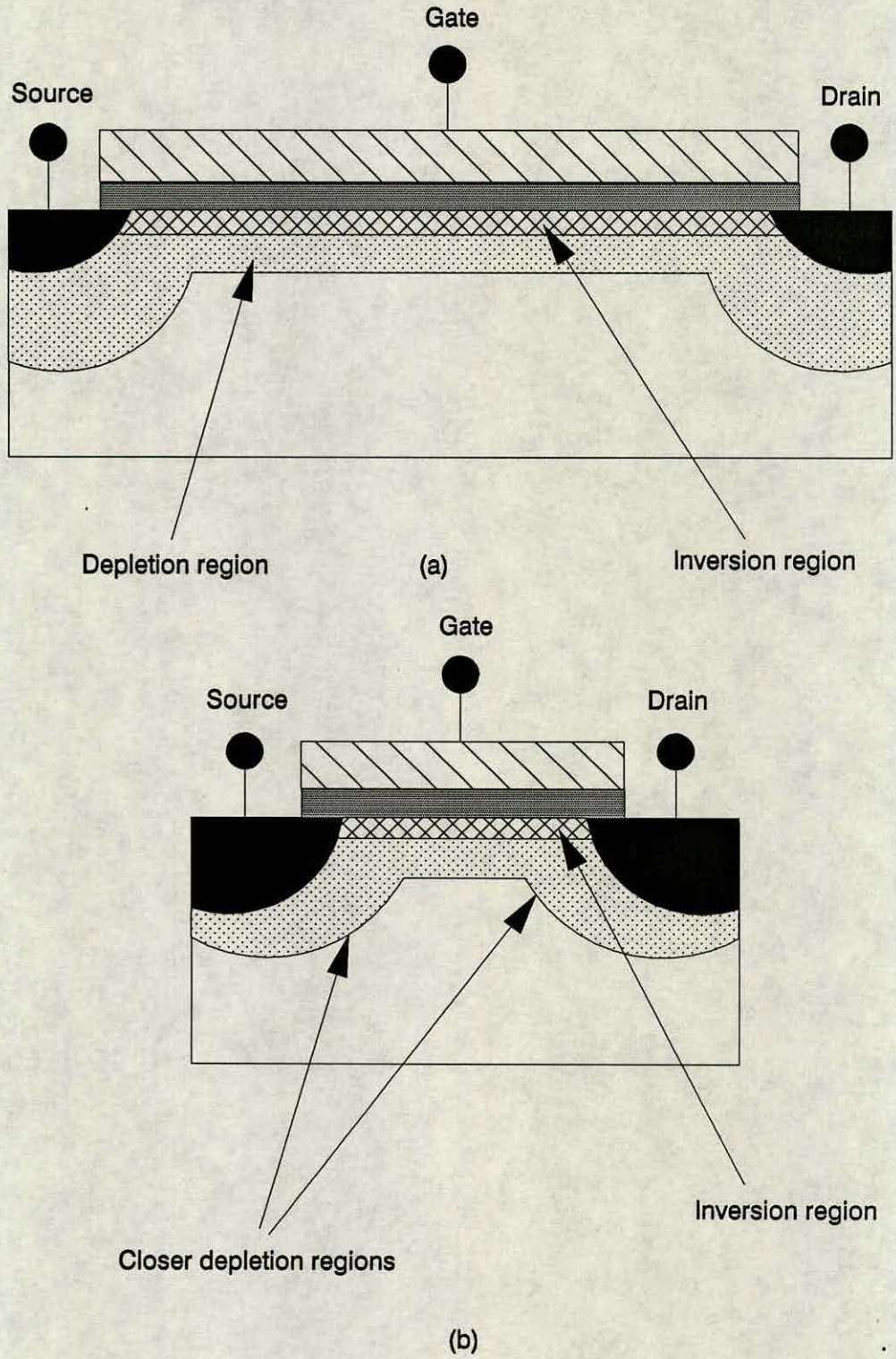


Figure 7–1: A schematic diagram showing depletion regions of (a) a long channel transistor and (b) short channel transistor

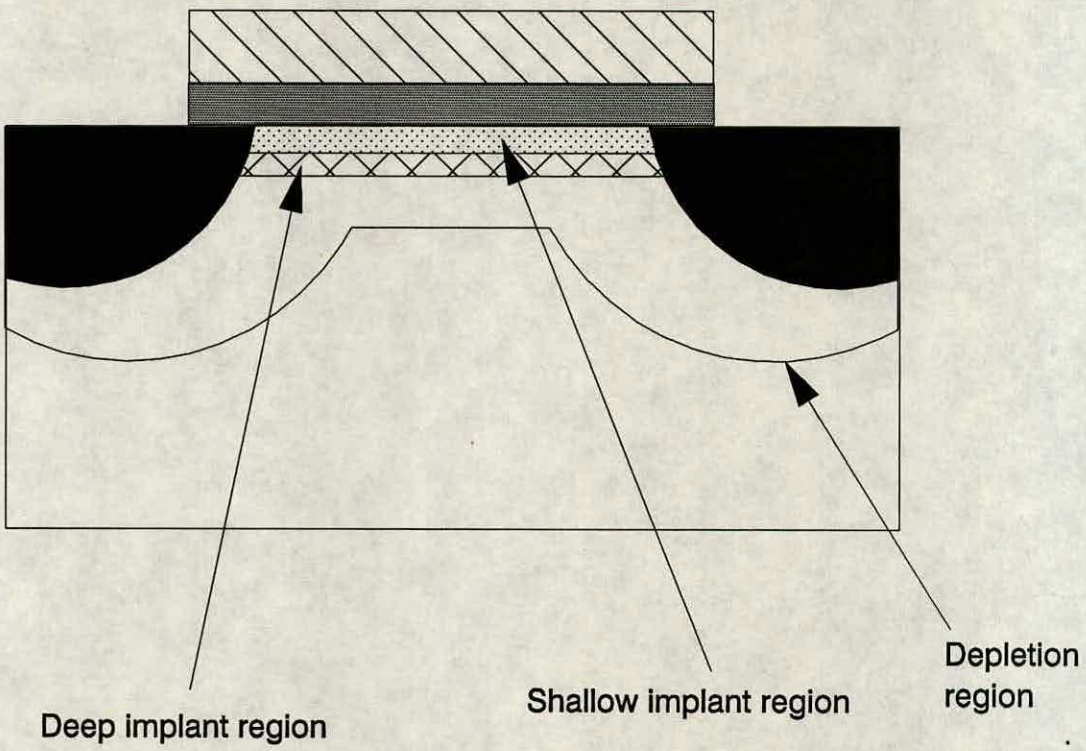


Figure 7–2: A schematic diagram showing the double implant structure and doping regions

- A shallower junction depth

It is the subject of this work to reduce the punchthrough voltage by increasing the substrate doping with the use of a double implant. The first implant is a low energy threshold voltage adjusting implant. The second implant is a higher energy implant and is used to increase the substrate doping below the surface, thus increasing the punchthrough voltage. The advantage of using an implant to increase the doping, rather than using a higher uniform substrate doping, is that the implant produces a localised high doping area. This is important as the higher the doping concentration, the higher the threshold voltage sensitivity (an undesirable effect). As a result of using the implant there will only be a small area of the device where this sensitivity is high [35].

Figure 7–2 shows the double implant structure and doping regions.

7.3.3 Avalanche breakdown

This form of breakdown is caused by the high electric fields present in short channel devices. Although also present in long channel devices, the effects are more prevalent in the short channel transistor. Figure 7–3 shows the flow of the carriers involved in avalanche breakdown. The high electric fields result in high velocity electrons which can generate electron-hole pairs due to implant ionization. These holes can then flow to the substrate (current 3), resulting in a substrate current. This current is used as a measure of the amount of electron-hole pairs generated due to impact ionization. Some of the holes generated can also be collected by the source (current 2), thus reducing the source barrier, allowing more electrons to be injected into the substrate (current 1). These electrons are attracted to the positive drain voltage and cause additional electron-hole pairs due to impact ionization, thus the whole process avalanches and leads to breakdown. Some of the electrons in the depletion region are also attracted to the positive gate voltage, resulting in a gate current (current 4). In this mode of operation the drain-channel-source is acting like a npn bipolar transistor.

The effect of doping on avalanche breakdown is the opposite to that for punchthrough ie a reduction in doping results in an increase of the breakdown voltage. This has resulted in the development of Lightly Doped Drain (LDD) devices to reduce avalanche breakdown in short channel devices.

For the given 1.5μm nMOS process, the choice of dose and energy must be large enough to increase punchthrough, but not too large to lead to an adverse effect on avalanche breakdown. There are still many combinations of dose and energy that result in an acceptable level of punchthrough or avalanche breakdown, but may not result in optimum values of other device parameters. These parameters may include those detailed earlier, namely threshold voltage, gamma and subthreshold swing.

The definition of breakdown (either punchthrough or avalanche) was taken to be the voltage required to produce 1nA of drain current, with the gate, source and substrate equal to zero volts. The nature of the breakdown can be determined by

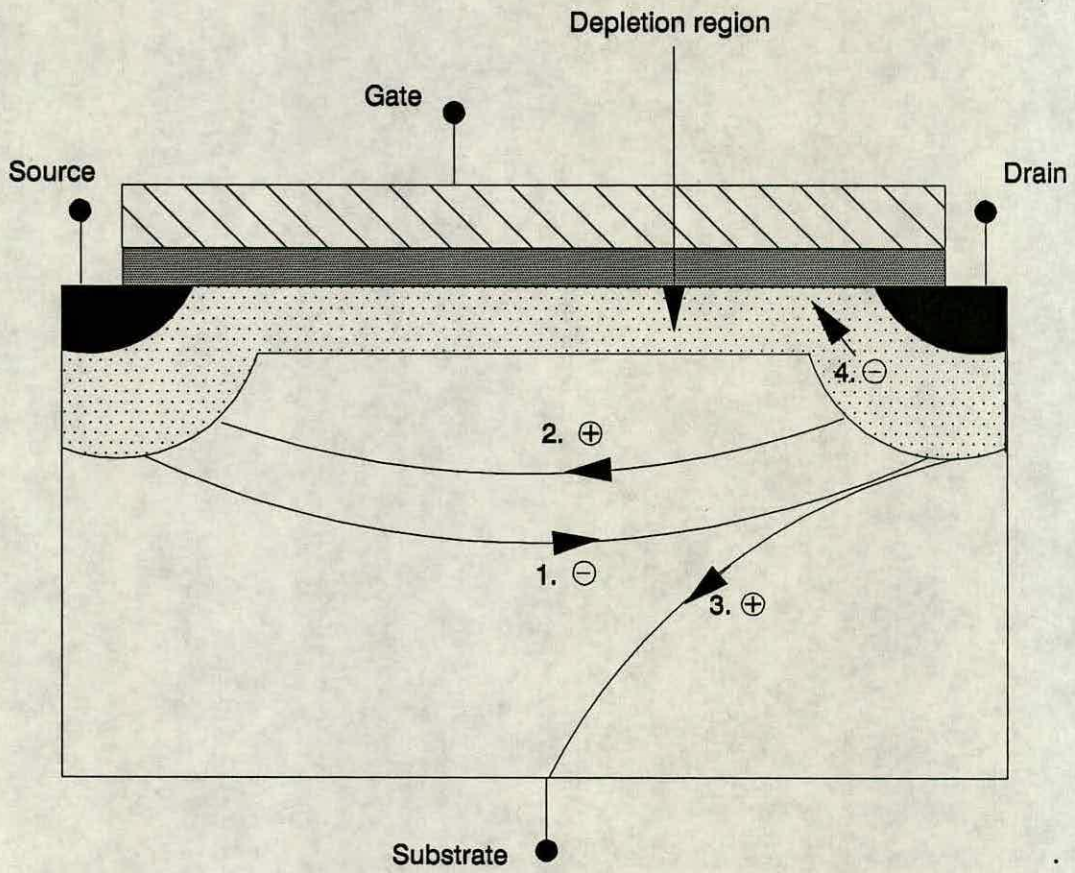


Figure 7–3: A schematic diagram showing the major flow of carriers in avalanche breakdown

the value of the substrate current. Simulation results show that for punchthrough the majority of the current flows from drain to source, whereas for avalanche breakdown the majority current flows from the drain to the substrate.

7.3.4 Gamma (body effect coefficient)

This is often referred to by different names. These include; gamma, the body effect coefficient, substrate effect or substrate sensitivity. A detailed analysis of the body effect is given in [108]. For the purposes of this work, gamma is considered to give a measure of the sensitivity of threshold voltage with a changing substrate bias. It can be considered qualitatively in the following manner. As the substrate bias is made more negative, this causes the negative space charge to increase, thus

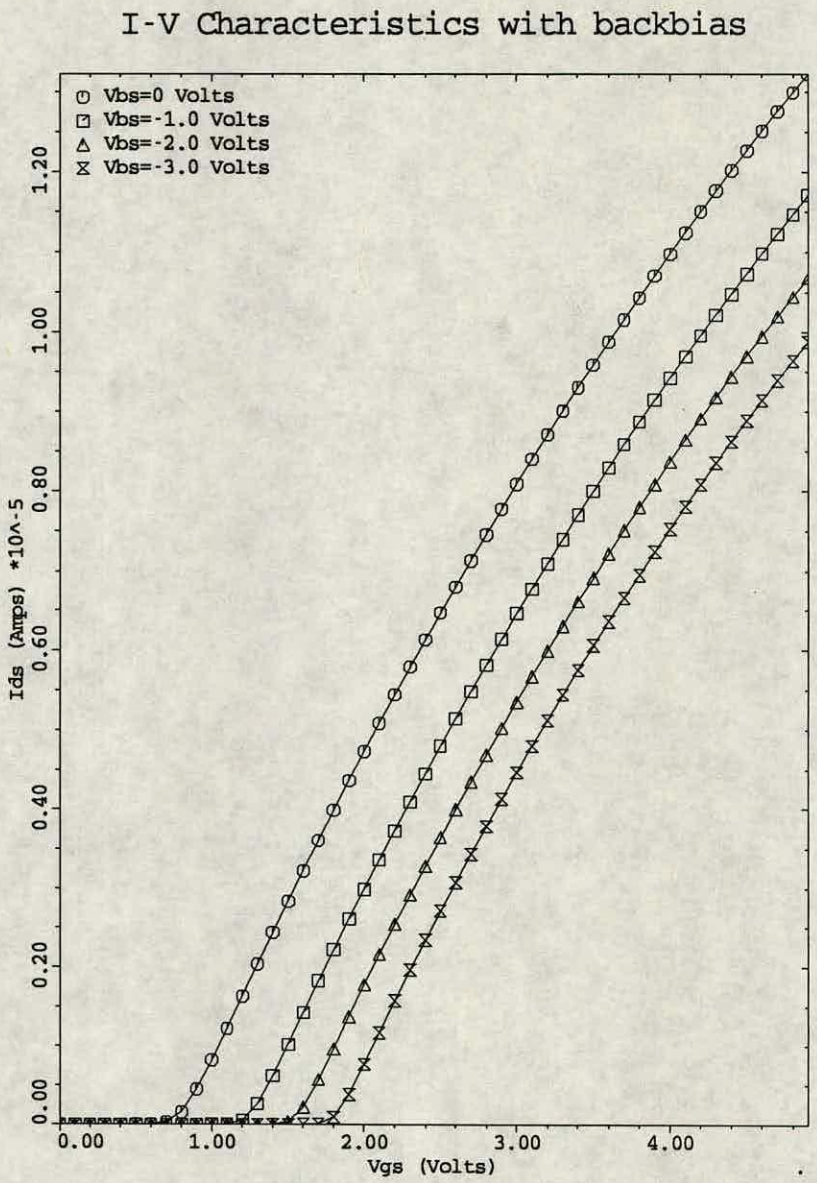


Figure 7–4: Effect of substrate bias on threshold voltage

the threshold must be increased further to compensate for this. A typical IV characteristic is outlined in figure 7–4, showing the effect of substrate bias on threshold voltage.

The definition for gamma will be taken as follows: [111]

$$\gamma = \frac{\sqrt{2q\epsilon_S N_a}}{C_o} \tag{7.2}$$

where ϵ_S is the permittivity of silicon, N_a is the substrate doping and C_o is the

capacitance per unit area and is defined as:

$$C_o = \frac{\epsilon_{ox}}{d_{ox}} \tag{7.3}$$

where ϵ_{ox} is the permittivity of oxide and d_{ox} is the oxide thickness.

Thus from equation 7.2 it is noted that gamma is only dependent on substrate doping and oxide thickness. For the subject matter of this work, oxide thickness is assumed constant, thus gamma will change depending on the substrate doping, which will vary depending on the implant dose and energy.

7.3.5 Subthreshold swing

The subthreshold region of a MOS transistor is when the applied gate voltage is below the threshold voltage. The drain current is then referred to as the subthreshold current, and the transistor is in the weak inversion area of operation. This is of particular interest in low voltage digital applications where the transistor is used as a switch, as the subthreshold region dictates how the device switches on and off. This is becoming important in sub-micron CMOS devices used in the portable consumer electronics market. As the subthreshold swing becomes smaller, so does the off-current or leakage current, thus increasing the battery life of products such as portable computers.

The subthreshold swing, S, is defined as the change required in the gate voltage to reduce the current by one decade [112].

$$S = ln10 \cdot \frac{dV_G}{d(lnI_d)} \tag{7.4}$$

Converting from ln to log_{10}, equation 7.4 becomes:

$$S = \frac{V_{G2} - V_{G1}}{log_{10}I_{D2} - log_{10}I_{D1}} \tag{7.5}$$

This corresponds to the reciprocal slope of the plot of ln Id versus V_G and is usually measured in mv/decade. Equations detailing the value of the subthreshold

current [59] show its dependence on, among other factors, the substrate doping. Thus the value of the implant will have an effect on the subthreshold current and hence the subthreshold swing.

7.3.6 Threshold voltage

Threshold voltage can be simply defined as the voltage required on the gate of the transistor to switch it on, ie when an appreciable level of drain current is flowing. With a zero back bias the threshold voltage, V_T can be simply defined as follows [112]:

$$V_T = V_{FB} + \phi_{si} - \frac{Q_B}{C_o} \tag{7.6}$$

where V_{FB} is the flat band voltage, ϕ_{si} is the surface band bending and $\frac{Q_B}{C_o}$ is the potential due to the space charge region.

The amount of charge beneath the gate is heavily dependent on the channel doping. Thus for the low energy threshold adjust implant, most of the implanted dopant will have a significant effect on the value of threshold voltage. For the deeper, higher energy implant, less of the implanted dopant will reside just below the surface, thus the effect on threshold voltage will not be as significant. There will, however, be a slight effect on the threshold voltage as the increased doping concentration will still influence the amount of charge beneath the gate.

For the purpose of this analysis, the threshold voltage,V_T was taken as the value when the tangent to the steepest part of the I_d versus V_G curve intercepts the horizontal axis, V_t minus half the drain voltage. Thus,

$$V_T = V_t - \frac{V_d}{2} \tag{7.7}$$

An example of the curve is shown in figure 7–5.

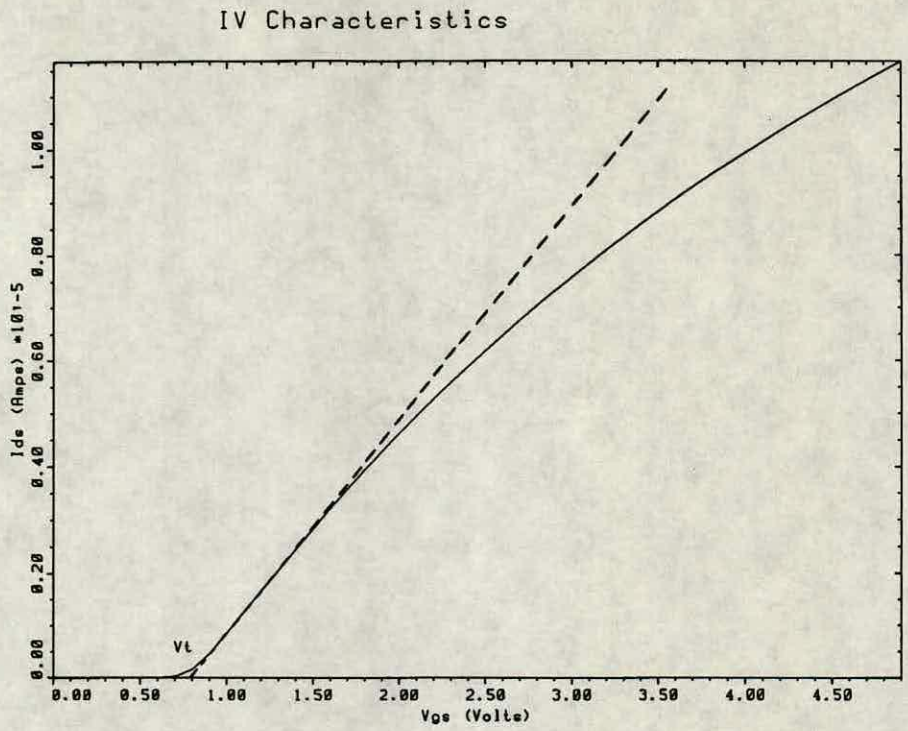


Figure 7–5: Calculation of threshold voltage

7.4 Defining the experiment

7.4.1 Response Surface Analysis

Initial screening experiments were performed using the RS/1 suite of software and RS Discover in particular. These initial experiments were to establish the correct range of operation for the implant conditions. The first range of dose and energy was taken to be 10^{10} to 10^{12} atoms cm^{-2} and 80 to 180 keV respectively. Simulations were performed at the specified experimental points and after entering the results for breakdown, gamma, subthreshold swing and threshold voltage back into RS/1 for statistical analysis, it was established that the specified range was too large. This was because it was not possible to adequately model the changes in breakdown voltage over this parameter space. After two more iterations through

THESIS - Experiment Worksheet

0	1 DOSE (atoms/cm2)	2 ENERGY (KeV)	3 BREAKDOWN (V)	4 REALBREAK (V)	5 GAMMA (V_half)
1	2e+11	180.00	5.80	6.00	0.514
2	4e+11	180.00	7.80	8.80	0.639
3	2e+11	80.00	6.40	6.00	0.461
4	2e+11	146.67	6.10	6.00	0.523
5	8e+11	146.67	11.60	11.50	0.800
6	8e+11	80.00	12.35	12.30	0.956
7	8e+11	113.33	12.30	12.50	0.890
8	6e+11	180.00	9.40	10.60	0.699
9	6e+11	80.00	12.00	11.00	0.864
10	4e+11	80.00	9.60	8.80	0.675
11	2e+11	113.33	6.20	6.00	0.508
12	8e+11	180.00	10.70	11.50	0.741

0	6 GAMMA_REAL (V_half)	7 SUB_THRES (mv/dec)	8 SUB_THRES_REAL (mv/dec)	9 THRESH (v)
1	0.497	94.62	88.90	0.5900
2	0.699	94.72	91.20	0.5975
3	0.478	95.90	92.30	0.6100
4	0.602	94.73	89.50	0.5975
5	0.767	95.66	95.86	0.6100
6	0.912	98.17	99.50	0.7200
7	0.888	97.61	97.40	0.6800
8	0.722	94.85	91.60	0.6025
9	0.866	98.14	97.90	0.7000
10	0.630	98.05	97.50	0.6800
11	0.538	95.04	91.30	0.6050
12	0.814	94.98	91.60	0.6025

1 DESTINATION
2 FACTORS
3 RESPONSES
4 PERF_STAT
5 MODEL
6 INCLUSIONS
7 CANDIDATES
8 DESIGN
9 WORKSHEET
10 SUMMARY
11 NEXT

More columns to display.
EXPER.OUTPUT>

Figure 7–6: The RS Discover worksheet for a D-optimal design

this screening process, the optimum range for the dose was taken as 2×10^{11} to 8×10^{11} atoms cm^{-2} and 80 to 180 keV for the energy. This range resulted in a good representation of the changes in breakdown voltage with different implant conditions.

These initial investigations also established that using a cubic model rather than the traditional quadratic model, resulted in a much more accurate fit. This was particularly the case for the breakdown voltage, where the cubic model was able to produce a more accurate representation of the simulation data.

The above ranges for dose and energy were input into RS/1 and a D-optimal design using a cubic model was chosen. The cubic model requires 4 factor levels in order to fit an equation to the data, resulting in 12 experimental runs. Figure 7–6 shows the suggested worksheet output from RS Discover for a D-optimal design.

For comparison purposes it was also decided to perform full-factorial analysis.

THESIS_CUBIC_FF - Experiment Worksheet

0	1 DOSE (atoms/cm2)	2 ENERGY (KeV)	3 BREAKDOWN (V)	4 REALBREAK (V)	5 GAMMA (V_half)
1	2e+11	80.00	6.40	6.00	0.461
2	4e+11	80.00	9.60	8.80	0.675
3	6e+11	80.00	12.00	11.00	0.864
4	8e+11	80.00	12.35	12.30	0.956
5	2e+11	113.33	6.20	6.00	0.508
6	4e+11	113.33	9.20	10.30	0.693
7	6e+11	113.33	11.50	10.50	0.816
8	8e+11	113.33	12.30	12.50	0.890
9	2e+11	146.67	6.10	6.00	0.523
10	4e+11	146.67	8.50	10.00	0.683
11	6e+11	146.67	10.40	10.40	0.749
12	8e+11	146.67	11.60	11.50	0.800
13	2e+11	180.00	5.80	6.00	0.514
14	4e+11	180.00	7.80	8.80	0.639
15	6e+11	180.00	9.40	10.60	0.699
16	8e+11	180.00	10.70	11.50	0.741

DESTINATION
2 FACTORS
3 RESPONSES
4 PERF_STAT
5 MODEL
6 INCLUSIONS
7 CANDIDATES
8 DESIGN
9 WORKSHEET
10 SUMMARY
11 NEXT

More columns to display.
EXPER.OUTPUT>

Figure 7–7: Part of the RS Discover worksheet for a full-factorial design

Figure 7–7 shows part of the required worksheet for a full-factorial design and a cubic model. For this design 16 runs are necessary.

The cubic model generated by RS Discover has the general form shown in equation 4.5, where x_1 and x_2 represent dose, D and energy, E, respectively. The terms in the model were transformed using the method described in equation 4.6 and have the following form:

$$\hat{D} = \frac{D - 5 \times 10^{11}}{3 \times 10^{11}} \tag{7.8}$$

$$\hat{E} = \frac{E - 130}{50} \tag{7.9}$$

Where $\hat{D}$ represents the transformed value of dose, with a midpoint of 5×10^{11} and the half range equal to 3×10^{11} atoms cm^{-2}. Energy, E, is similar with a midpoint of 130 and a half range of 50keV.

This transformation ensures that all factor settings are within the range $-1 \leq 0 \leq 1$, thus allowing the relative importance of each term to be evaluated.

Simulations were then performed at the full-factorial points and also at the existing operation point, namely a dose of 7×10^{11} atoms cm^{-2} and an energy of 140 keV. Wafers were also fabricated at the above points, with two wafers included for each experimental point. This is required in case of breakages to wafers and also to average out any wafer to wafer variations. The results obtained for the real device measurements are outlined in chapter 8.

It should be noted at this stage, however, that due to a processing error, the source/drain implant dose for the real devices was not equal to the value detailed on the runsheet, namely 7×10^{15} atoms cm^{-2}. The actual value implanted was determined using experimentation and then simulation. Firstly the sheet resistance of the source/drain area was measured using two different methods.

- A four point probe
- A Greek cross

A four point probe was used to establish the sheet resistance of the batch test wafer. This was calculated at approximately 300 Ω/square. By forcing a current through the Greek cross test structure and measuring the voltage the sheet resistance was calculated. Again this resulted in a value of approximately 300 Ω/square.

The one dimensional simulator, SUPREM3, was then used to establish the implant dose required to produce a sheet resistance of 300 Ω/square. The value taken was 2.5×10^{14} atoms cm^{-2}.

After further analysis and comparison with the results obtained for the original dose of 7×10^{15} it was confirmed that the breakdown characteristics for a 1.5 μm channel using the original dose were similar to those of a 1 μm channel with the new dose of 2.5×10^{14} atoms cm^{-2}. Thus it was now possible to obtain useful results by analysing (both in simulation and reality) a 1μm channel transistor.

7.4.2 Taguchi analysis

Taguchi analysis was performed, for simulation only, on the 1.5 μm nMOS process. This facilitated a comparison between RSM and Taguchi techniques. Existing in house software was used to carry out the Taguchi analysis [113]. Extra code was added to this to allow the use of *larger is better* and *smaller is better* signal-to-noise ratios. This was necessary as the original objectives were to maximise breakdown voltage and to minimise gamma and subthreshold swing. The original code had only provision for *target is best* analysis.

In order to restrict the number of runs to a comparable level to that of RSM analysis, it was decided to vary the dose and energy at two factor levels and two noise levels. The factor levels were 80 and 180 keV and 2×10^{11} and 8×10^{11} atoms cm^{-2}. The noise levels were chosen at $\pm 5\%$ of the nominal value. This resulted in 16 runs, the same as that for the full-factorial RSM technique. Figure 7–8 shows the Taguchi array for this design and table 7–1 shows the corresponding values of dose and energy and the resulting values for breakdown, threshold voltage, gamma and subthreshold swing.

The signal-to-noise ratio and mean analysis for the Taguchi results will be given later in the chapter. This allows easy comparison between RSM and Taguchi simulation results.

7.5 Simulation Procedure

This section details the simulation procedure and the results obtained from the simulation of a 1μm channel transistor.

The simulation procedure is as follows:

- Perform process simulations using the 2D simulation package, SUPREM4.
- Load results from the process simulation into the device simulation program, PISCES -IIB.

Noise factors

			4	3	2	1	trial / factor
			1	1	2	2	D
Control factor levels			1	2	1	2	E
trial / factor	D	E					
1	1	1	●	●	●	●	
2	1	2	●	●	●	●	
3	2	1	●	●	●	●	
4	2	2	●	●	●	●	

Figure 7–8: The Taguchi array showing the different control factor levels and noise factors

- Use PISCES to establish breakdown voltage, threshold voltage and sub-threshold swing.

- Generate in PISCES a series of I_d vs V_{gs} curves for different values of substrate bias.

- Load the above curves into TOPEX to extract a value for γ.

Using the software system outlined in chapter 6, the necessary simulation run files were generated from the initial RS Discover worksheet file, containing the required settings for dose and energy. Each simulation file was placed in a different directory and then SUPREM4 was executed on each drive file.

The next section deals with the process simulation aspects in simulating the 1μm channel device.

Dose (atoms cm^{-2})	Energy (keV)	V_{bd}(V)	$\gamma(V^{\frac{1}{2}})$	S_t (mv/dec)	V_t(V)
1.9e11	76	6.3	0.4502	96.93	0.611
1.9e11	84	6.2	0.4577	97.32	0.609
2.1e11	76	6.6	0.4647	96.8	0.612
2.1e11	84	6.5	0.4705	97.15	0.61
1.9e11	171	5.8	0.5078	95.01	0.596
1.9e11	189	5.7	0.5015	94.87	0.596
2.1e11	171	6.0	0.5239	95.05	0.5995
2.1e11	189	5.9	0.5174	94.9	0.596
7.6e11	76	12.4	0.9430	99.49	0.7125
7.6e11	84	12.4	0.9303	99.04	0.705
8.4e11	76	12.375	0.9806	98.84	0.715
8.4e11	84	12.375	0.9650	99.71	0.7075
7.6e11	171	10.7	0.7383	96.23	0.6045
7.6e11	189	10.2	0.7142	95.76	0.601
8.4e11	171	11.1	0.7534	96.38	0.6055
8.4e11	189	10.6	0.7277	95.87	0.601

Table 7–1: Table showing Taguchi simulation results

7.5.1 SUPREM4 Analysis

The essential goal in this stage of the analysis is to obtain an accurate and representative doping profile of the source/drain regions and also the channel doping. Without this, realistic results from device simulation would be made difficult, if not impossible.

One of the most important criterion in arriving at an accurate solution for process simulation is the provision of an adequate initial grid. The grid was chosen so as to be finer in the area close to the silicon surface (up to about 0.5μm) and not so fine further down in the substrate. The grid used is shown in figure 7–9.

There is, of course, a trade off between the size of the grid and the time taken

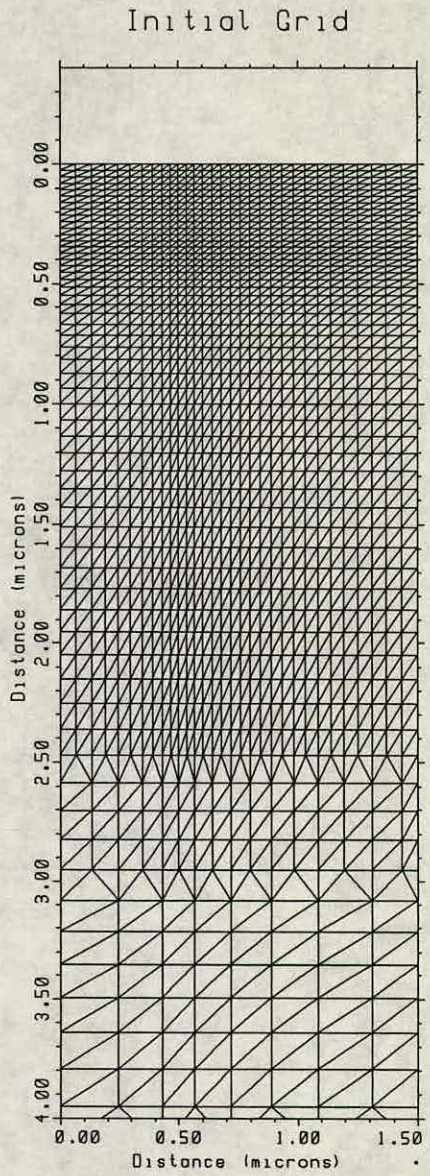


Figure 7–9: The initial grid used in the SUPREM4 simulation

for the simulation to be completed. After extensive investigations of different grid structures it was decided that this grid gave good results, with not too excessive computation times.

In a further effort to reduce the simulation time, the runsheet was studied in detail and only those steps which were considered to significantly affect the final simulation results were included. When considering the effects of breakdown, subthreshold swing, threshold voltage and gamma, the areas of most interest are the source/drain areas and the channel region. Thus care was taken to ensure that

the doping profiles were as accurate as possible in these regions. Also, since the device is symmetrical only half of the transistor was simulated and then flipped about a vertical axis at X = 0 at the end of the simulation. This has the obvious benefit of reducing the number of nodes by a factor of 2, hence reducing the required simulation time.

The following figures illustrate how the boron concentration in the channel varies with the two parts of the double implant and also with high diffusion temperature steps. Figure 7–10 shows the boron concentration in the channel region after the first part of the double implant. This corresponds to a dose of 7×10^{11} atoms cm^{-2} and an energy of 25 keV. Due to the low energy the boron concentration is high close to the surface, hence the large effect on the threshold voltage.

Figure 7–11 details the channel boron concentration after the second part of the double implant. This implant had a dose of 7×10^{11} atoms cm^{-2} and an energy of 140 keV. The higher energy results in a peak in the concentration deeper down into the substrate.

Figure 7–12 shows the final doping profile in the channel at the end of the simulation procedure. Due to the high temperature diffusion steps the impurity has diffused, showing a more smooth profile.

As mentioned previously, also of great importance for accurate device characteristics is the source/drain profile. The breakdown voltage is particularly affected by this profile and by the amount of sideways or lateral diffusion in particular.

Figure 7–13 shows the doping profile for the amended dose of 2.5×10^{14} atoms cm^{-2} immediately after the arsenic source/drain implant and before any high temperature steps.

Figure 7–14 shows the same doping profile at the end of the simulation procedure. The effective channel length has been reduced due to the lateral diffusion. This is why for small geometry processes it is essential to keep high temperature diffusion steps to a minimum, so as the effective channel length is as large as possible.

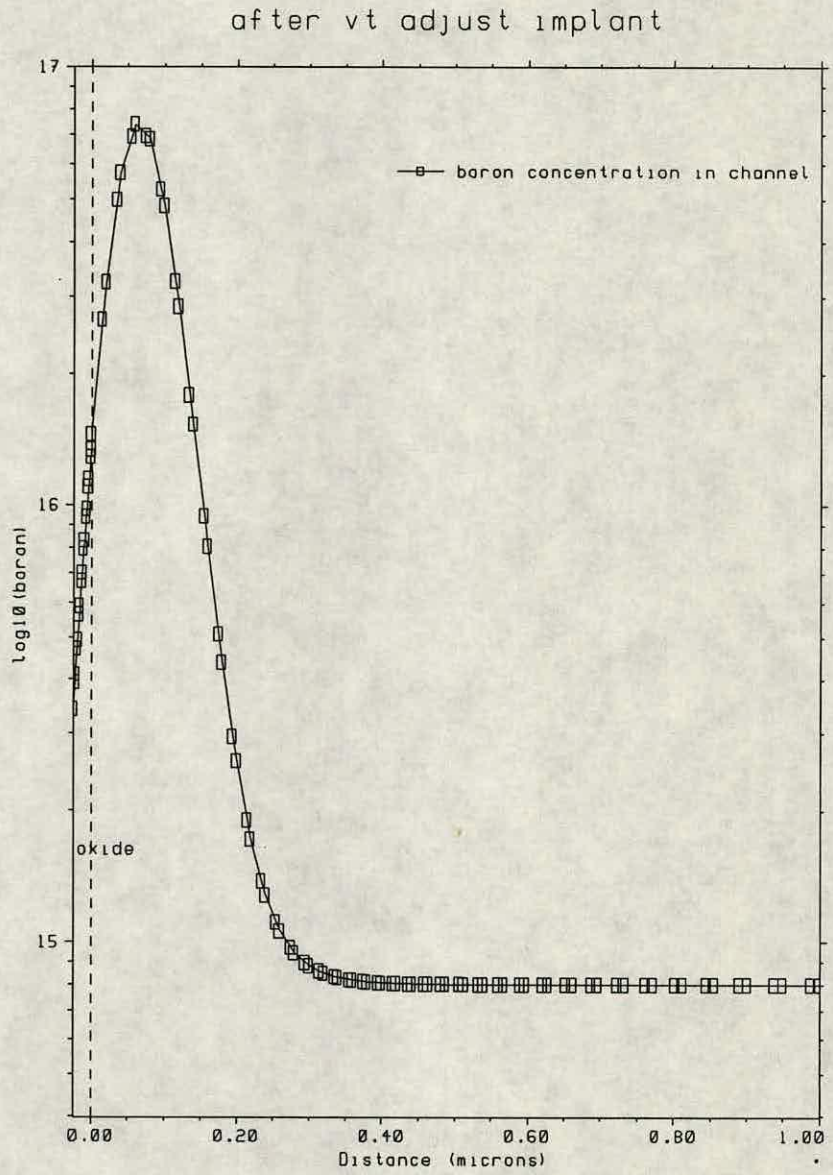


Figure 7–10: Channel region boron concentration after the first part of the double implant

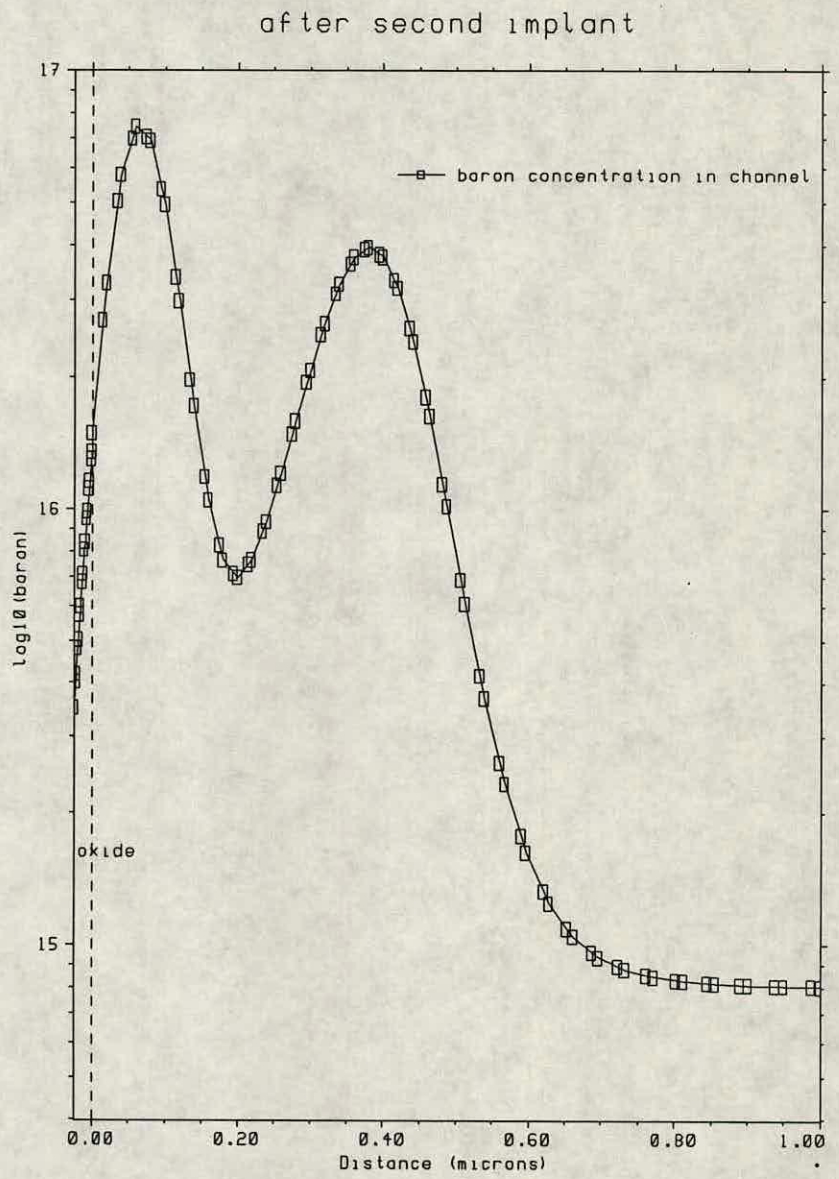


Figure 7–11: Channel region boron concentration after the second part of the double implant

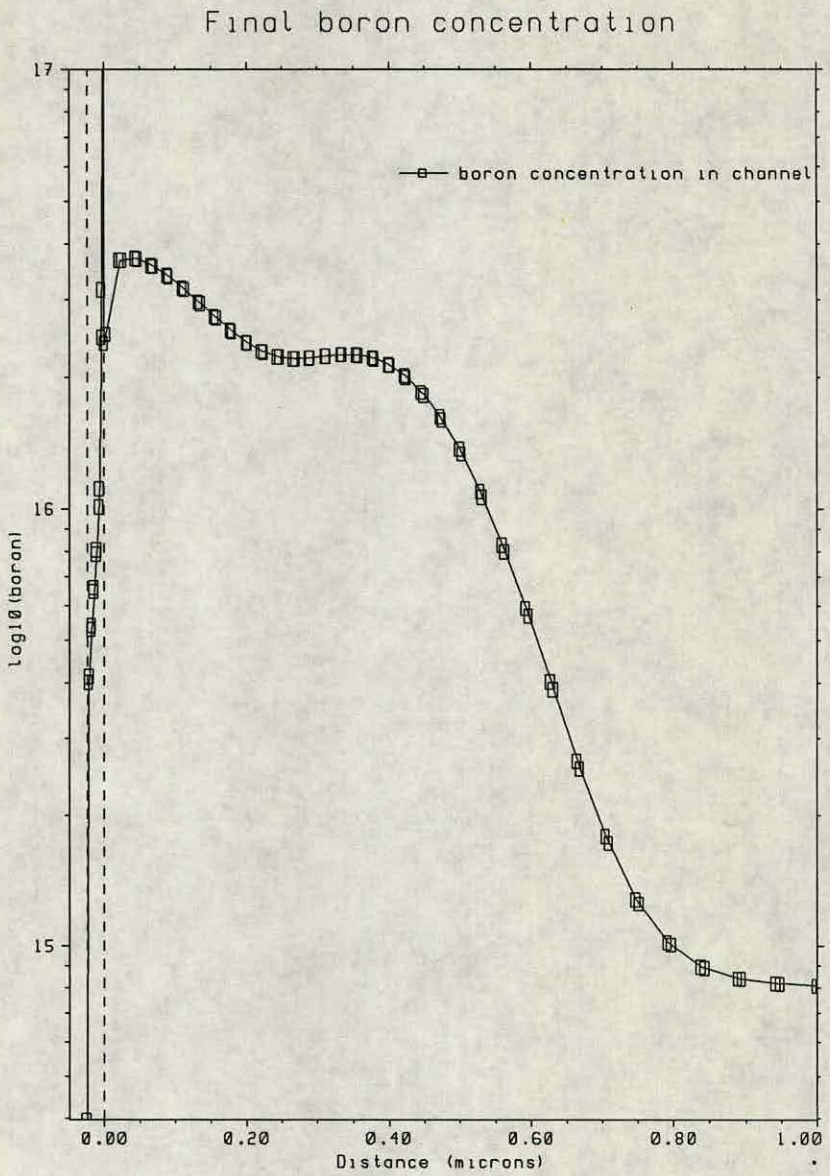


Figure 7–12: Final boron concentration after all processing steps were simulated

For comparison purposes figure 7–15 shows the final source/drain profile for the original value of dose, ie 7×10^{15} atoms cm^{-2}. As can be seen from the figure, the lateral diffusion is greater than that for figure 7–14. Thus the breakdown voltage is increased for the smaller value of dose, hence the reason why a $1\mu m$ channel with a low dose has similar breakdown characteristics to a $1.5\mu m$ channel with a higher dose.

Figure 7–16 details the method used to determine the junction depth of the n-type region. The junction is defined as when the n-type arsenic and p-type boron

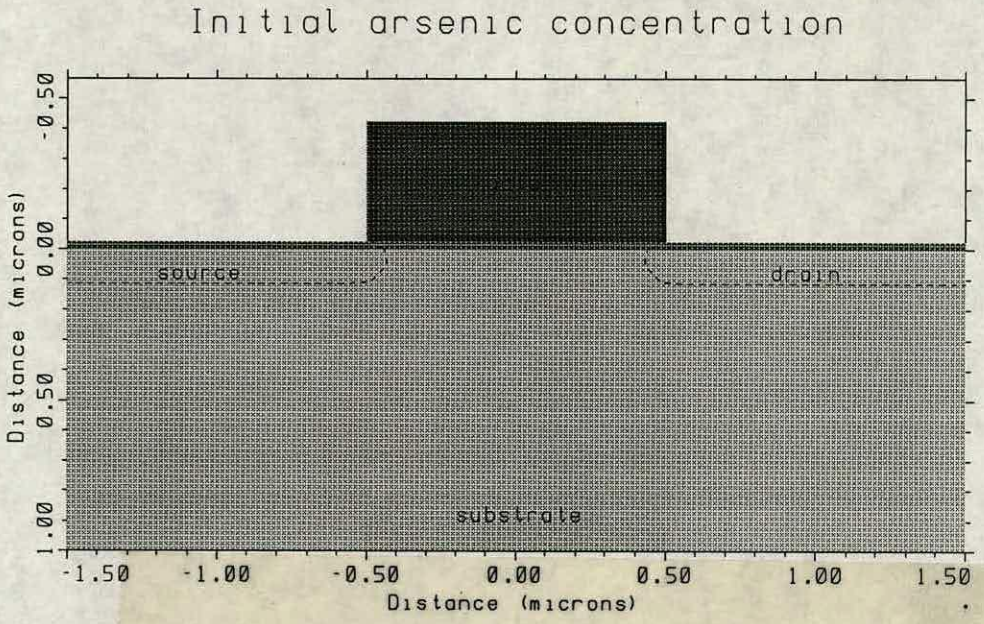


Figure 7–13: **Position of the junction depth immediately after the source/drain implant**

concentrations are equal, ie the intersection of the two lines in figure 7–16. For this situation, the junction depth is approximately equal to 0.185 μm.

The final structure and doping profiles for the simulation is shown in figure 7–17.

7.5.2 Device analysis

The above profile from SUPREM4 was then loaded into PISCES-2B to perform device analysis. There were three device simulations that were required to establish the values of the four responses; breakdown, gamma, subthreshold swing and threshold voltage. The same simulation results can be used to calculate subthreshold swing and threshold voltage.

Each of the three simulations performed will now be outlined in more detail.

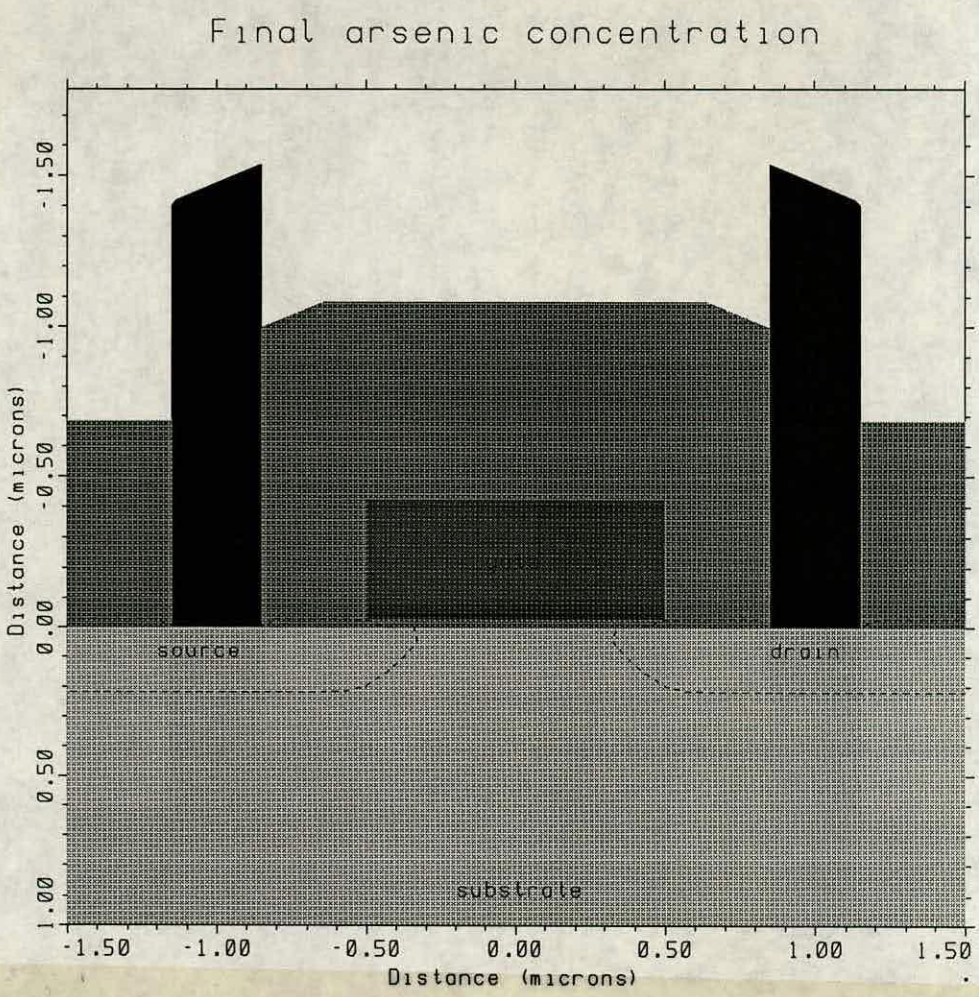


Figure 7 Position of the junction depth after all processing steps were completed

7.5.3 Breakdown voltage simulation

The condition for this simulation was zero volts on the gate, source and substrate and the drain was then ramped up until 1nA of drain current flows. Care should be taken to ensure that the depth of the device structure is large enough to accommodate the increased depletion region due to the high drain voltage. The nature of the breakdown (either punchthrough or avalanche) depends on the value of dose and energy. It was found that the higher the dose and the lower the energy the more the likelihood of avalanche breakdown. The opposite was the case for punchthrough, ie a low dose and a high energy. A typical curve for breakdown analysis is detailed in figure 7–18.

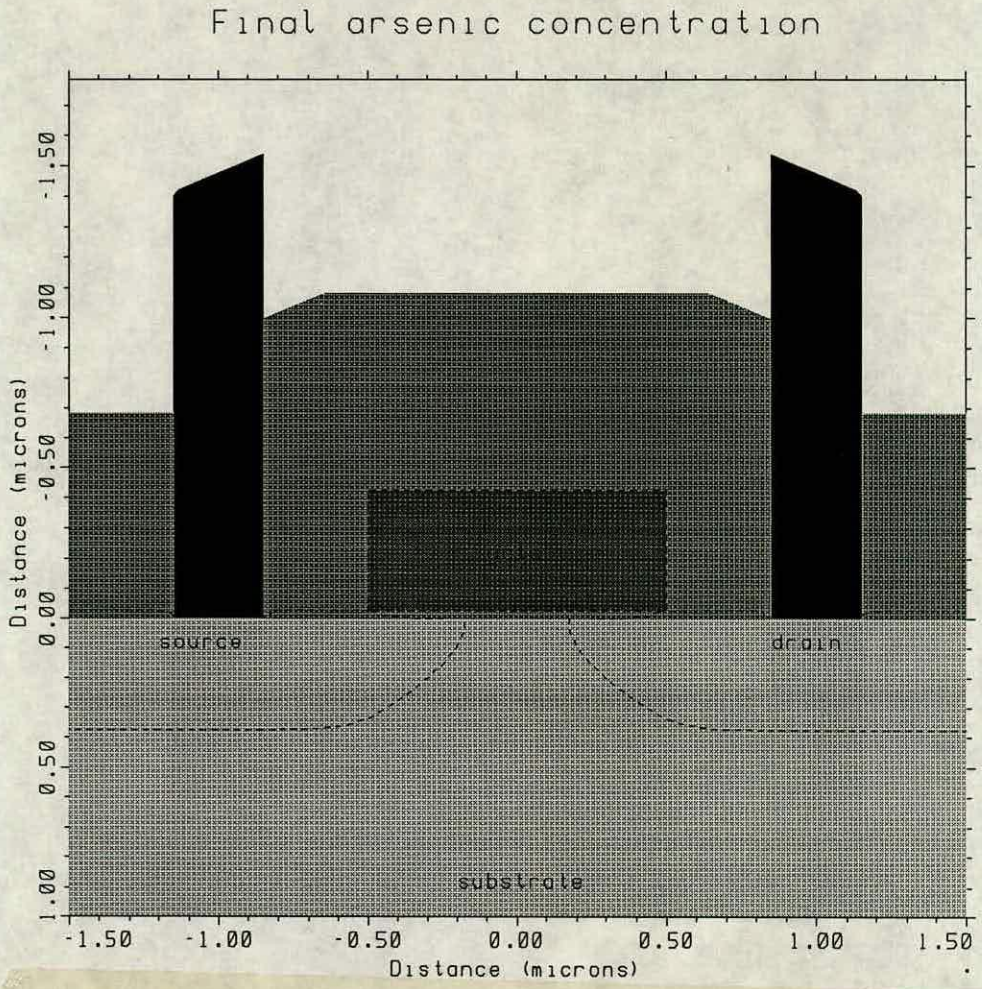


Figure 7- Final junction depth position for the original dose

The actual breakdown values obtained for the different simulated points are given in table 7–2.

7.5.4 Gamma simulation

Since the value of gamma was to be extracted using TOPEX, it was necessary to generate a set of I_d v V_{gs} curves in PISCES-2B to include in TOPEX.

The conditions for this simulation were; drain voltage = 0.1V, source voltage = 0V and the gate voltage was ramped up from 0 to 5V in steps of 0.1V. This was carried out for substrate bias voltages of 0 to -3V. The different IV curves for

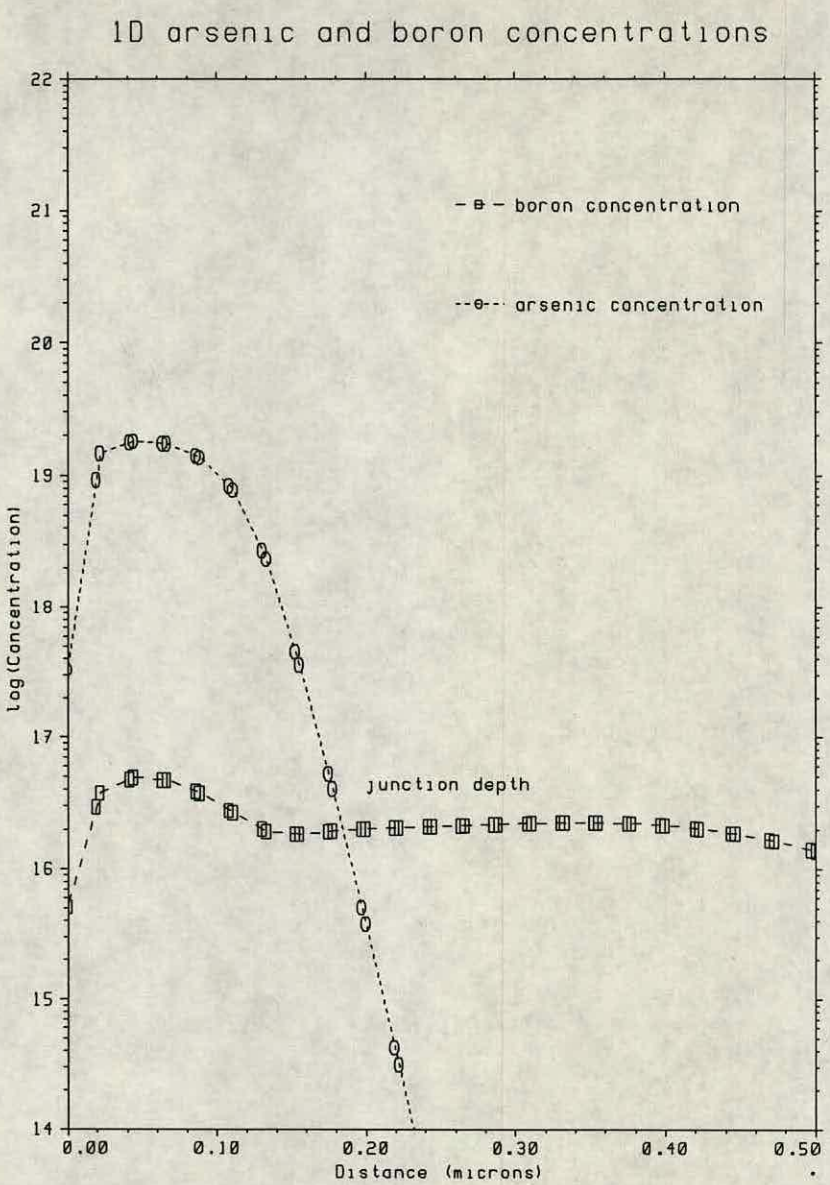


Figure 7–16: Arsenic and boron concentration intersection showing junction depth

the specified experimental points were then loaded into TOPEX for extraction of gamma. The values of gamma extracted are detailed in table 7–2.

The typical IV curves obtained from the device analysis were similar to that shown in figure 7–4.

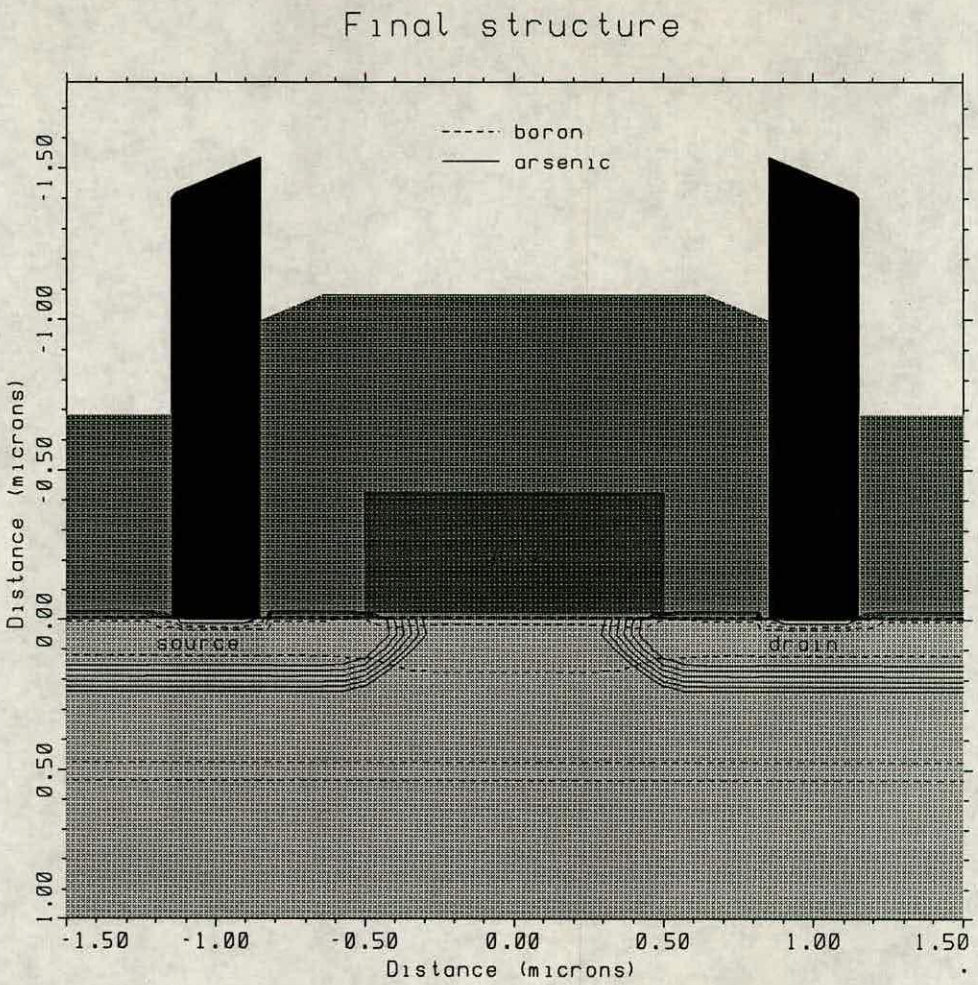


Figure 7–17: The complete simulation structure

7.5.5 Threshold and subthreshold simulation

The conditions for this simulation were the same as that for the gamma simulation, except that the gate voltage was ramped up in steps of 0.05V up to a value of 2V. The value of the substrate bias voltage was set to 0V. The value of threshold voltage was calculated as described in section 7.3.6. The value of subthreshold swing was calculated using PISCES. The results obtained for the threshold and subthreshold simulations are given in table 7–2.

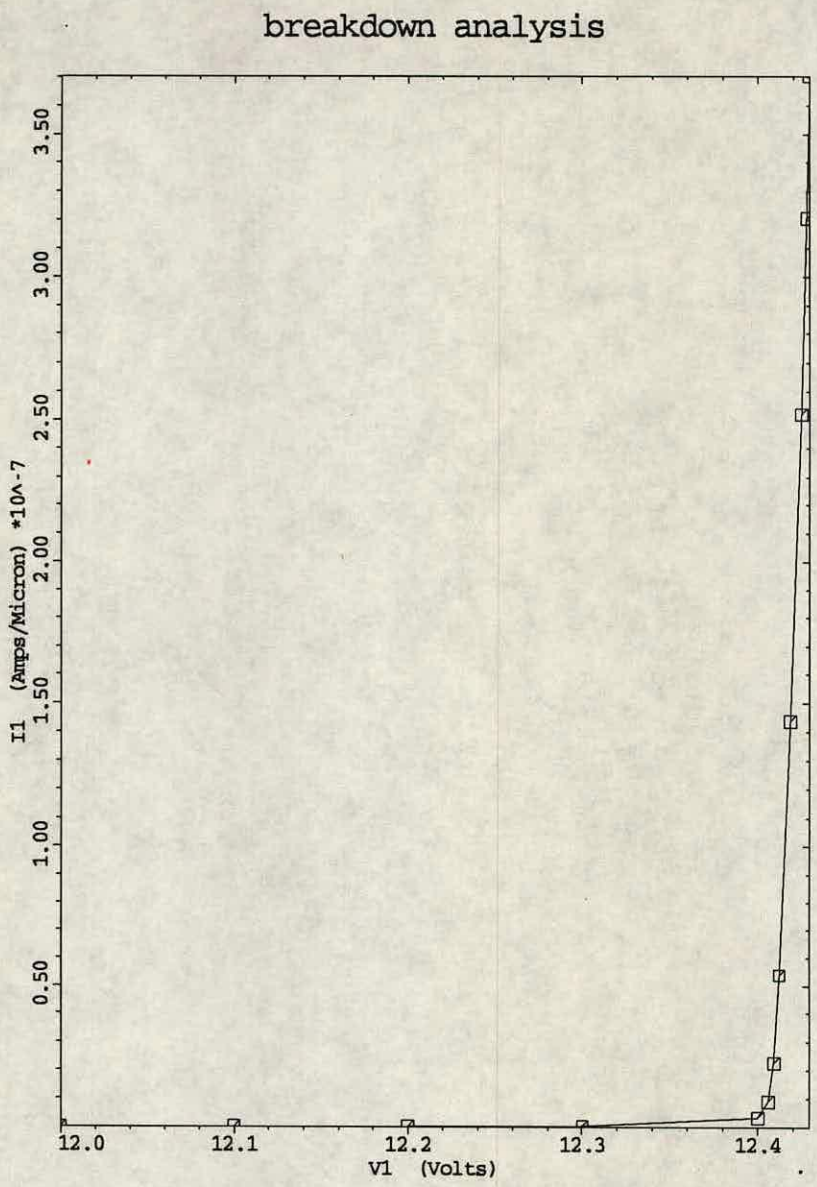


Figure 7–18: A typical curve showing breakdown

7.6 Analysis of results

These sections deal with the results obtained from the simulated data. An analysis of the real results and comparison with simulation is given in chapter 8.

Dose /atoms cm^{-2}	Energy /keV	V_{bd}/V	$\gamma/V^{\frac{1}{2}}$	S_t / mv/dec	V_t/V
2e11	80	6.4	0.461	95.9	0.61
4e11	80	9.6	0.675	98.05	0.68
6e11	80	12	0.864	98.14	0.70
8e11	80	12.35	0.956	98.17	0.72
2e11	113	6.2	0.508	95.04	0.605
4e11	113	9.2	0.693	95.82	0.61
6e11	113	11.5	0.816	96.72	0.615
8e11	113	12.3	0.890	97.61	0.68
2e11	147	6.1	0.523	94.73	0.5975
4e11	147	8.5	0.683	95	0.6025
6e11	147	10.4	0.749	95.31	0.605
8e11	147	11.6	0.800	95.66	0.61
2e11	180	5.8	0.514	94.62	0.59
4e11	180	7.8	0.639	94.72	0.5975
6e11	180	9.4	0.699	94.85	0.6025
8e11	180	10.7	0.741	94.98	0.6025

Table 7–2: Table showing simulation results

7.6.1 RSM analysis

The results from the simulations were loaded into RS/1, a model was fitted and analysis of the D-optimal design performed. The first stage was to check that the models fit the data adequately. A good measure of fit is given by the R-squared error. Table 7–3 shows the R-squared errors obtained for the different responses.

The '*' indicates that the response has been transformed, eg to a reciprocal, square root etc. RS/1 recommends whether a transformation should be made or not, and if so, which transformation to perform. In this situation only the results for breakdown and gamma indicated that a transformation was necessary. An R-squared value greater than 0.95 was considered as an acceptable fit.

Response	R-sq error
breakdown	0.9993*
gamma	0.9987*
subthreshold	0.9852
threshold	0.9879

Table 7–3: R-squared errors for the fitting of models for simulation

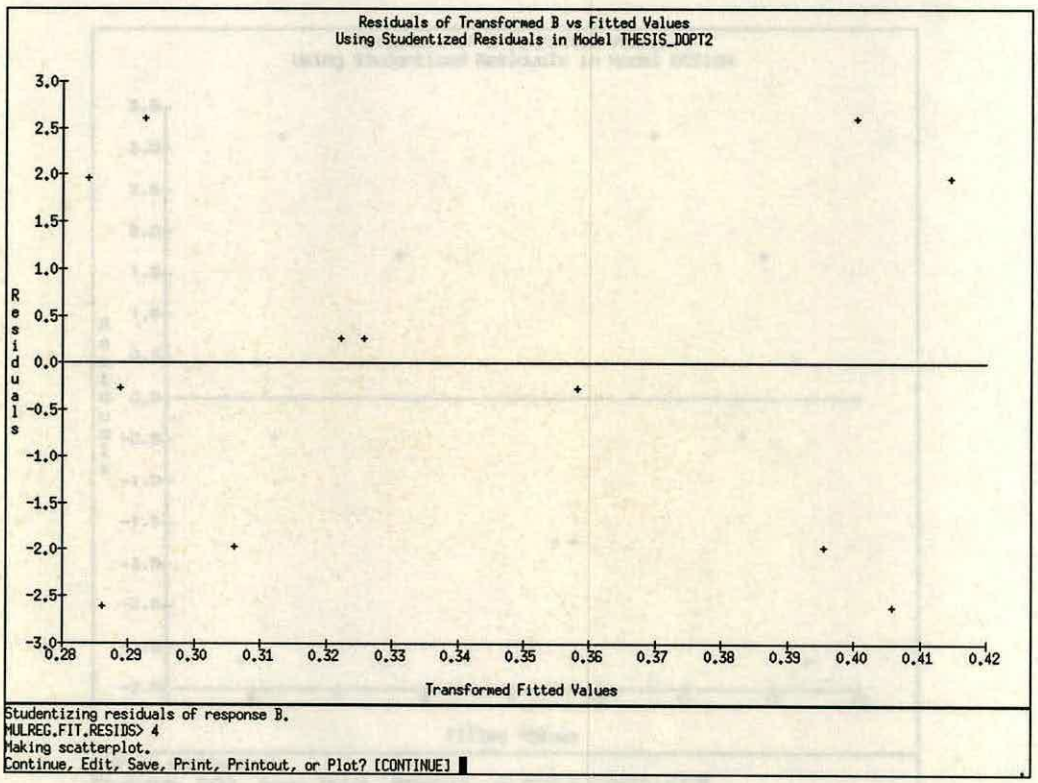


Figure 7–19: Scatterplot of simulation breakdown residuals ($\frac{1}{sqrt}$ transformation)

The residuals table for each response was examined and an example is shown in figure 7–19. This compares the difference between the simulated response and the model response, for each operating condition. A non-random form indicates a lack of model fit. Similar plots were also generated for gamma, subthreshold swing and threshold voltage.

Analysis of the coefficients and ANOVA tables also showed good agreement

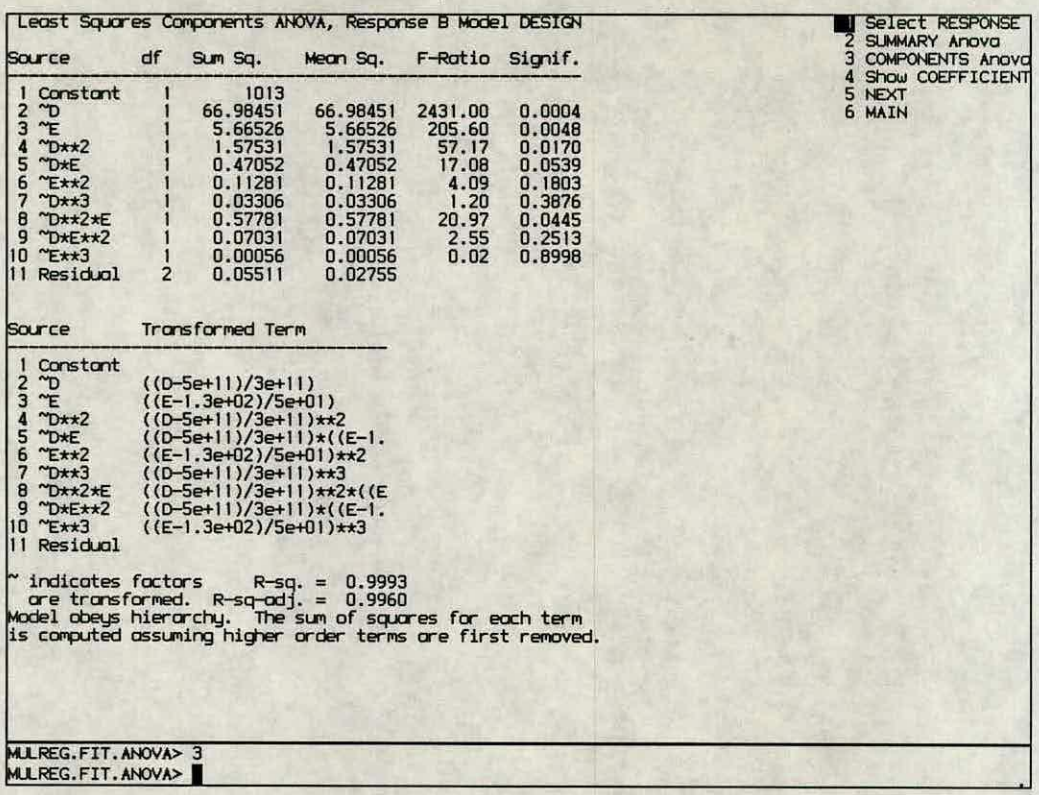


Figure 7–20: ANOVA table for simulated breakdown voltage.

between the model and the actual data values. From these tables it was possible to ascertain which terms contribute most to the accuracy of the model. The ANOVA table for breakdown voltage is shown in figure 7–20. The smaller the significance term the larger the contribution of that term in the model. Thus the linear term in D has the largest model contribution for breakdown voltage.

Terms with significance > 0.05 can be ignored.

Contourplots

This section outlines the different contourplots obtained from RS/1 for simulated data. Figures 7–21, 7–22, 7–23 and 7–24 describe the results obtained for the D-optimal experiment for breakdown, gamma, subthreshold swing and threshold voltage respectively.

Breakdown was divided into three areas, less than 9V, 9V - 10V and greater than 10V. Less than 9V was considered unsatisfactory, 9V - 10V marginal, and greater than 10V satisfactory. These areas are detailed in figure 7–21.

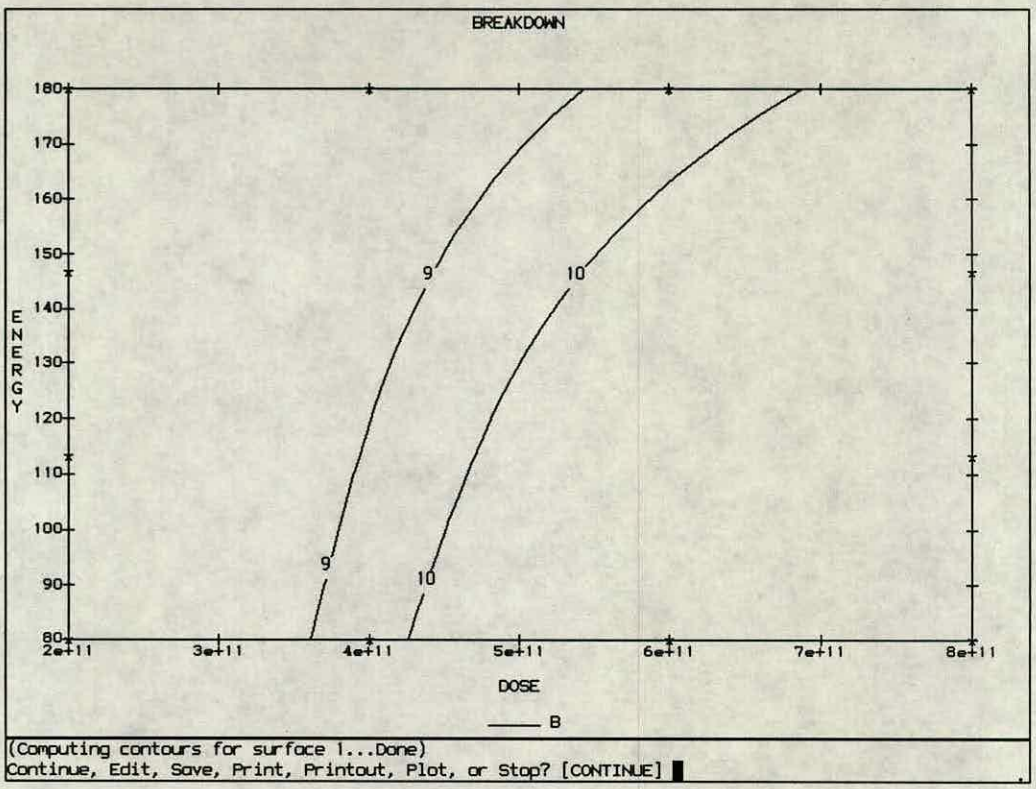


Figure 7–21: A contourplot of simulated data for breakdown analysis

The next stage of the analysis is to find the optimum operating conditions. This procedure will be outlined in the following section.

7.6.2 Optimum operating conditions

In order to establish the optimum operating conditions it was necessary to define a mathematical expression which related all the responses of interest to a single composite function. This expression was termed the composite response, CR, and is of the general form given in equation 7.10, where *weight* refers to a weighting term and $f()$, refers to a function.

$$CR = (weight)f(V_{bd}) + (weight)f(\gamma) + (weight)f(S_t) \qquad (7.10)$$

It should be noted that because threshold voltage was defined as being kept constant $\pm$ 0.075V, it was not included in the definition of CR. Two definitions of

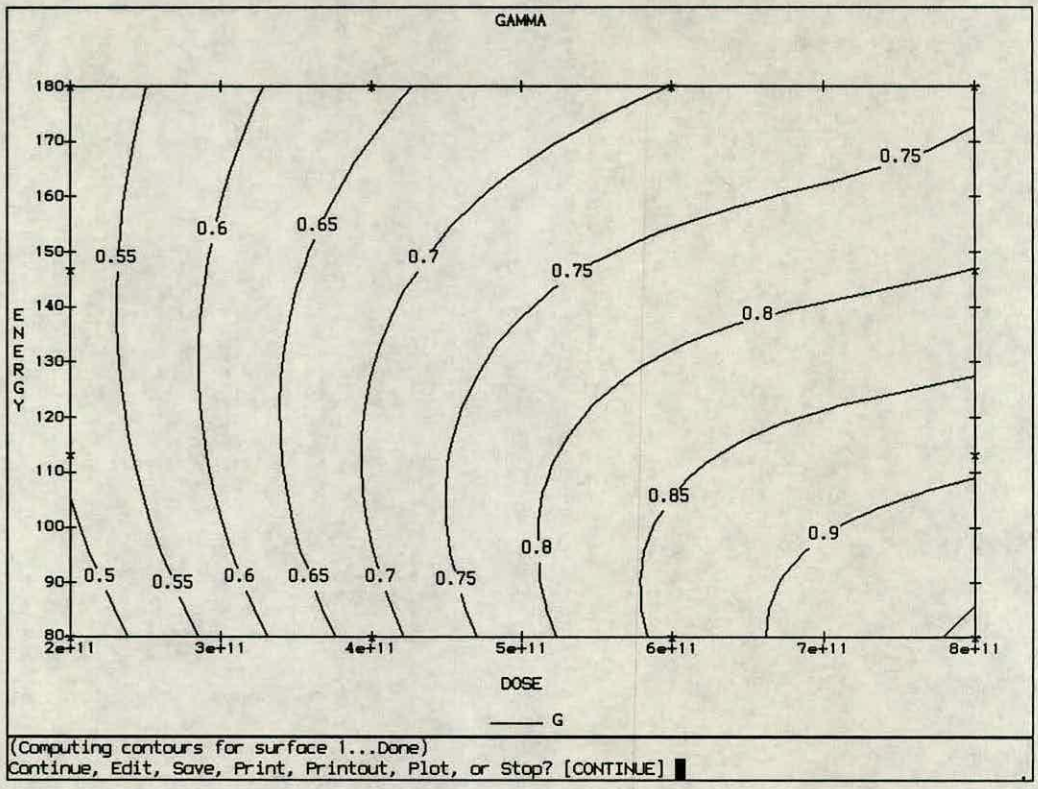


Figure 7–22: A contourplot of simulated data for gamma analysis

composite response were used in the analysis. The first definition of the composite response is described in equation 7.11.

$$CR1 = \frac{\gamma}{mean_{\gamma}} + \frac{mean_{V_{bd}}}{V_{bd}} + \frac{S_t}{mean_{S_t}} \tag{7.11}$$

Where *mean* refers to the average or mean value of the response in question. The equation was defined to make $CR1$ a minimum. A brief analysis of the equation shows that $CR1$ will be a minimum when γ and S_t are minimum and when V_{bd} is maximum, something which is consistent with the original specification.

This formula was then input into RS/1 using the `define formula` option and contours were plotted showing the composite response, CR1 and the limiting values for breakdown and threshold voltage. Figure 7–25 shows the plots obtained for the simulation data.

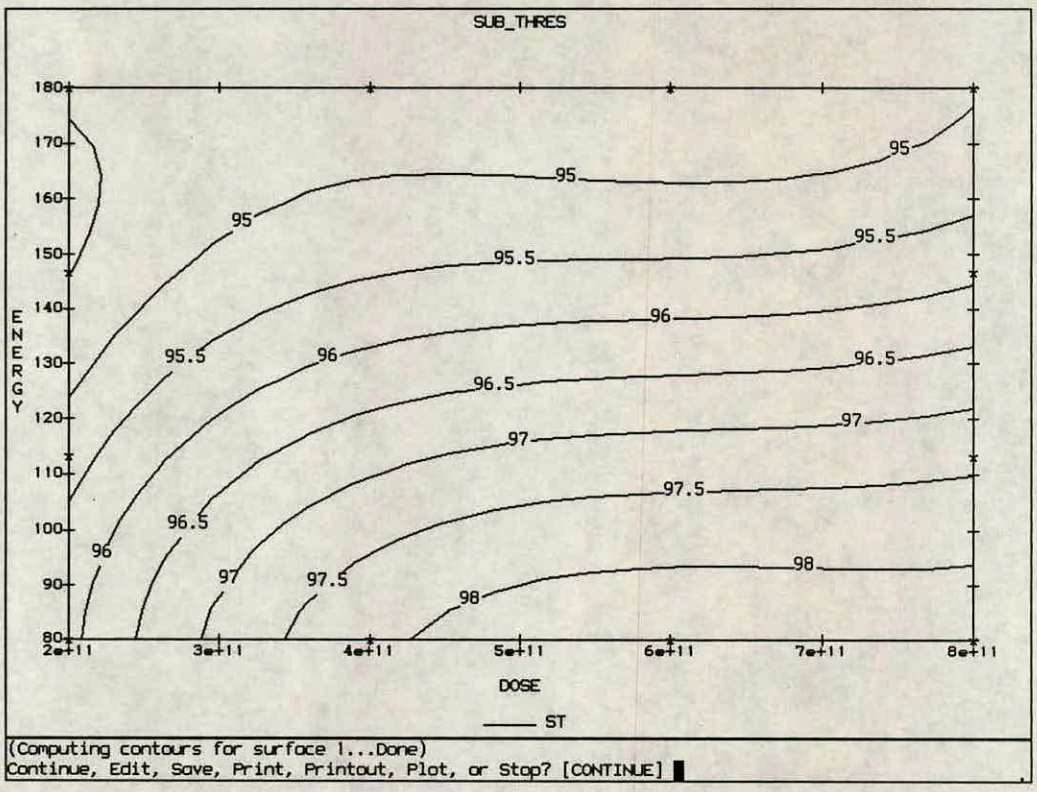


Figure 7–23: A contourplot of simulated data for subthreshold analysis

Analysis of figure 7–25 shows that the minimum value of CR1 (given breakdown and threshold voltage constraints) is at a value of dose of 8×10^{11} and an energy of 180keV, this is consistent with a qualitative analysis of the raw data.

There is, however, a problem with the definition of the composite response, CR1. No account is made for artificial weights placed on the different terms in the equation. This is due to the fact that the responses have not been normalised about their centre points and thus each term does not contribute equally to the overall value of the composite response.

Due to the inadequacies of the above approach, the values of the responses were normalised about their centre points, leading to the second definition of composite response. This expression is outlined below in equation 7.12 for the responses in question and is based on that defined previously in equation 4.6. It is defined as being optimised for its minimum value.

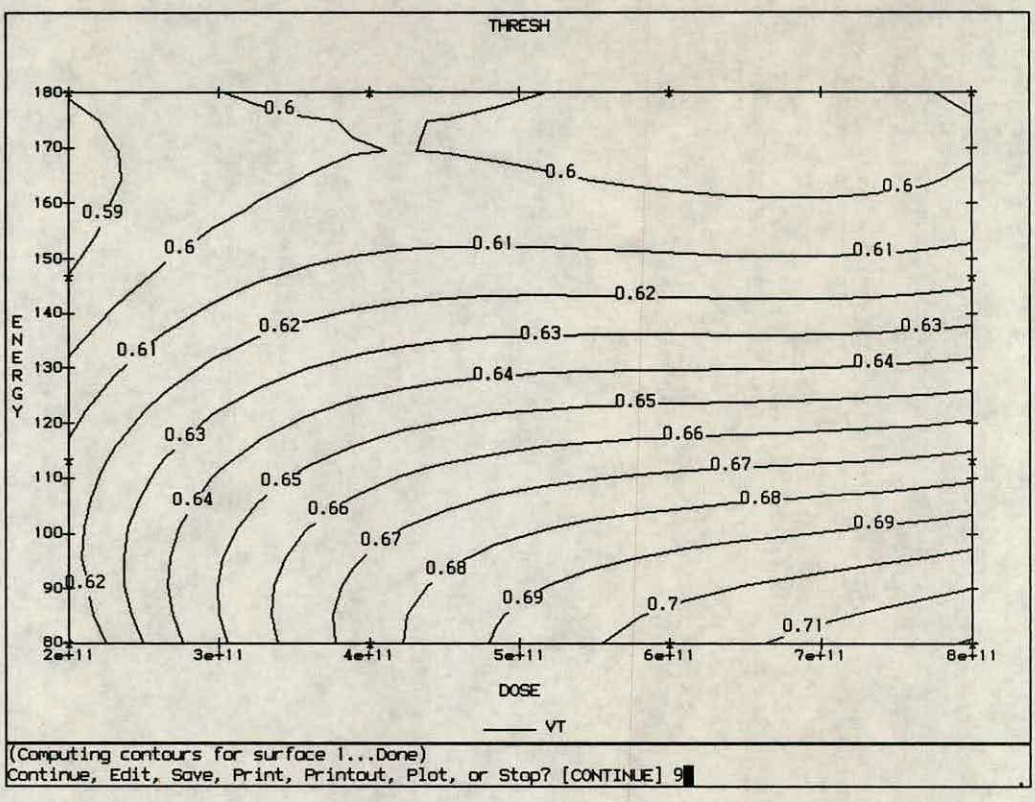


Figure 7–24: A contourplot of simulated data for threshold analysis

$$CR2 = (\frac{\gamma - median}{\frac{1}{2}range}) - (\frac{V_{bd} - median}{\frac{1}{2}range}) + (\frac{S_t - median}{\frac{1}{2}range}) \qquad (7.12)$$

Each term in the expression (relating to a particular response) can only vary in the range -1 $\leq 0 \leq$ 1, thus any weights placed on the terms in the equation are real in their effect.

A brief study of equation 7.12 will show that CR2 will be minimum when γ and S_t are minimum and V_{bd} is maximum, thus satisfying the original design criteria. This formula was defined in RS/1 and contours were plotted for CR2, along with the limits for breakdown and threshold voltage. Figure 7–26 shows the plot obtained for the simulated data.

The optimum operating point for the simulated data was extracted from RS/1, given the constraints of breakdown and threshold voltage. The value of dose was

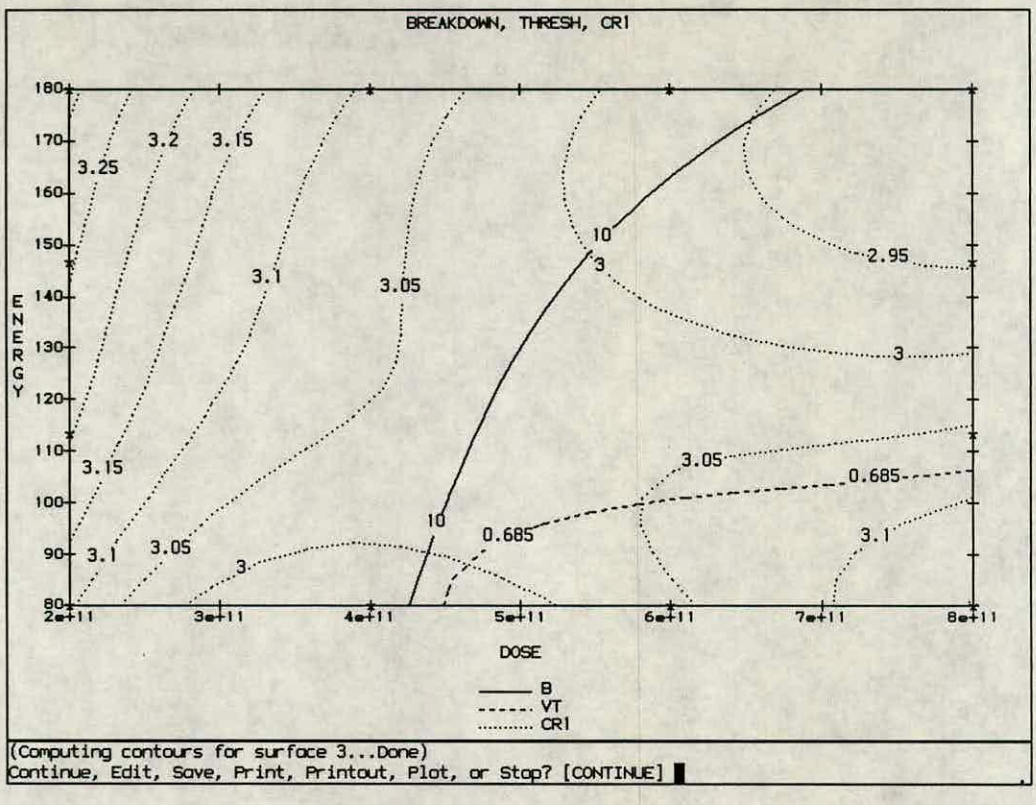


Figure 7–25: Contourplots for simulated data, defining the optimum region for the first definition of composite response

7.25×10^{11} and an energy of 180 keV. This agrees well with a qualitative analysis of the raw simulation data.

Additional investigation was performed to establish the effects of removing a term from the composite response expression. Since the breakdown voltage needs only to be greater than 10V, it was removed from the CR2 expression. This results in a trade off between γ and S_t only. It would be expected that removing the breakdown voltage term from the expression would have little or no effect on the optimum operating point, since this point would still be considered optimum when only considering γ and S_t. This is verified in figure 7–27 which shows the contourplots for CR2 without a term for breakdown.

The plot is similar to that for figure 7–26, which includes breakdown in the expression. In addition the optimum operating point is essentially in the same area of the parameter space.

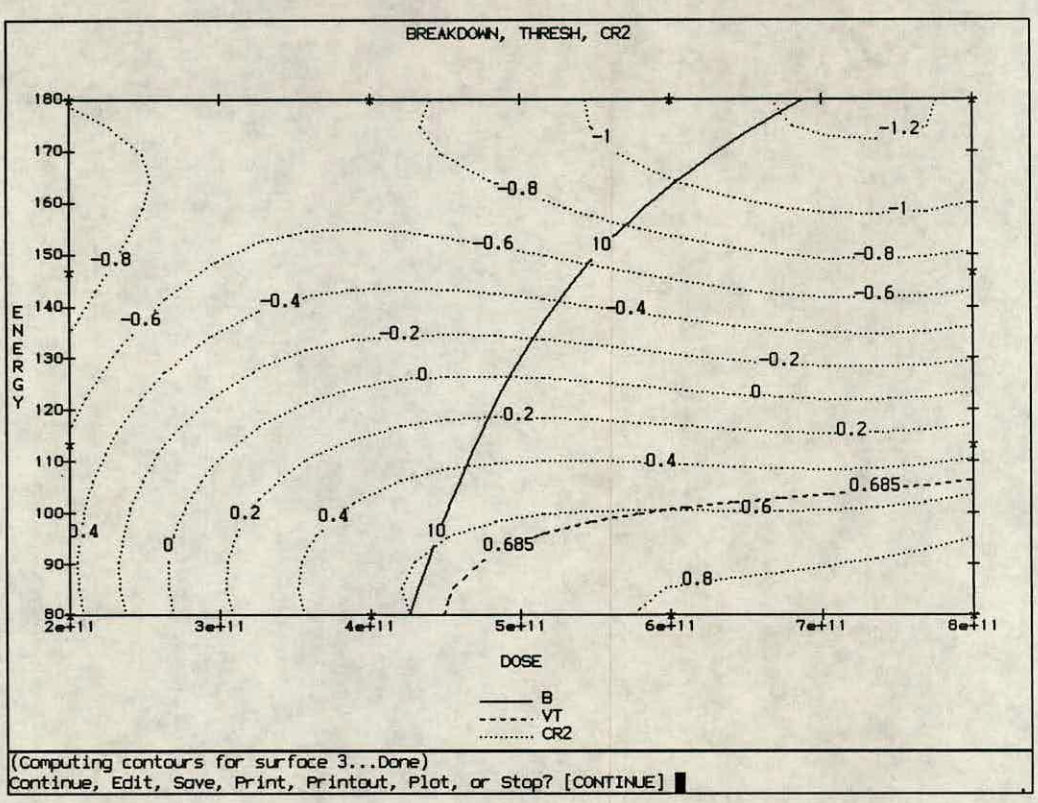


Figure 7–26: Contourplots for simulated data, defining the optimum region for the second definition of composite response

7.6.3 Sensitivity analysis

Now that the optimum point has been established, the question that needs to be asked is, "How sensitive is this point to variations in the control factors?" Tightening the distribution of device characteristics is essential in improving the quality of the finished product. A smaller sensitivity of the device parameters to variations in the input control factors will result in an increase in the yield of the product in the manufacturing environment. This section will address the issues involved in reducing the sensitivities of the device characteristics. Essentially it involves being prepared to sacrifice a little on device performance, in order to reduce the manufacturing sensitivity.

The general principle behind the analysis is to investigate the gradient of the contours. The closer the contours are to each other, the more sensitive the response will be to input variations. Partial derivatives were used to gain a measure

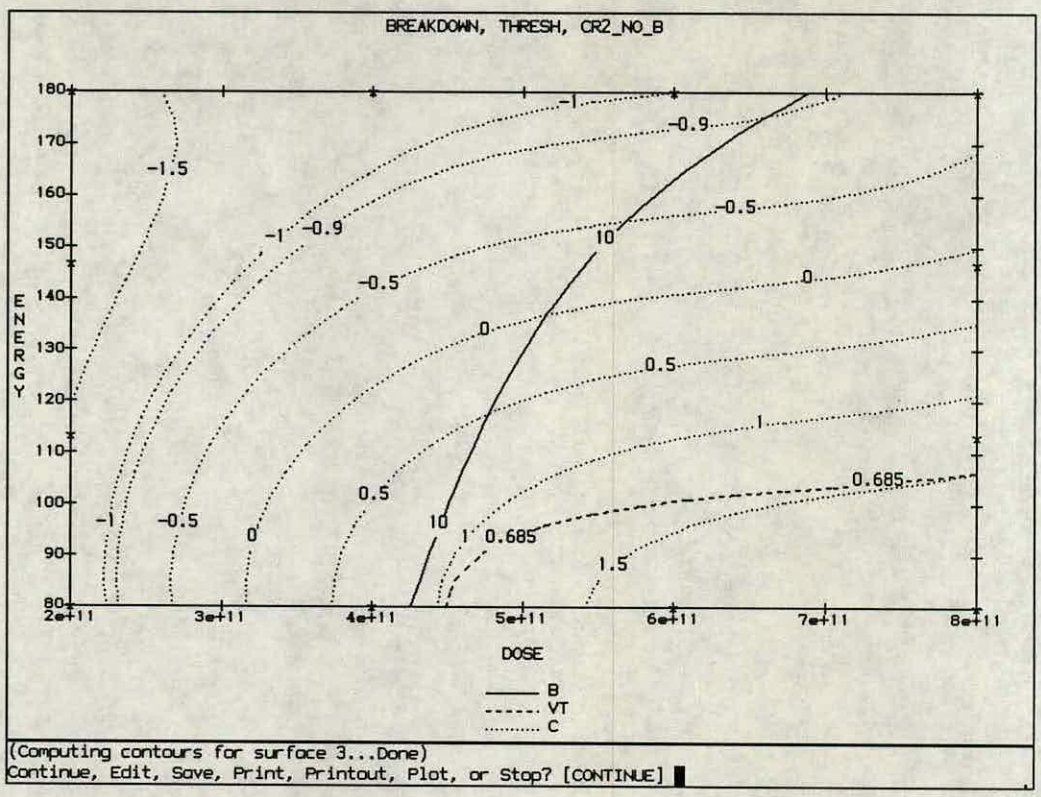


Figure 7–27: Contourplots for simulated data, defining the optimum region for the second definition of composite response without a term for breakdown voltage

of the steepness of the slope of the contours. Equation 7.13 shows the formula expression used to calculate the new composite response, with sensitivities included for breakdown voltage. The new response, CR2SENS is made up of the old composite response, CR2, and extra terms for the partial derivatives of breakdown voltage with respect to dose and energy.

$$CR2SENS = CR2 + \frac{1}{2}[(\frac{\partial V_{bd}}{\partial D})^2 + (\frac{\partial V_{bd}}{\partial E})^2] \tag{7.13}$$

The partial derivatives are calculated in RS/1 using the `$DERIVATIVE` function. As was the case for the responses, the values for the partial derivatives were normalised about their centre point, to ensure that each term contributes equally to the overall response. Each derivative then varies between -1 and +1, and for this reason the terms are squared to ensure positive values. As a result, each

derivative term can now vary between 0 and 1. Thus the both terms in total can vary between 0 and 2. It was because of this that the weight $\frac{1}{2}$ was placed on the derivatives, so that the total variation can be only between 0 and 1. This means that the contribution of the derivatives is comparable to that of CR2 itself. The value of the weighting factor depends on the importance placed upon the sensitivities.

When considering the sensitivities due to all four responses, a similar expression to that of equation 7.13 was obtained. Instead of having two terms for the partial derivatives, there were eight; two for each of the four responses. The weighting factor was also changed from $\frac{1}{2}$ to $\frac{1}{8}$, so that the total possible variation was still only from 0 to 1. This expression is outlined in equation 7.14.

$$CR2SENSALL = CR2 + \frac{1}{8}[(\frac{\partial V_{bd}}{\partial D})^2 + (\frac{\partial V_{bd}}{\partial E})^2 + (\frac{\partial \gamma}{\partial D})^2 + (\frac{\partial \gamma}{\partial E})^2 + (\frac{\partial S_t}{\partial D})^2 + (\frac{\partial S_t}{\partial E})^2 + (\frac{\partial V_{th}}{\partial D})^2 + (\frac{\partial V_{th}}{\partial E})^2] \quad (7.14)$$

Figure 7–28 shows the composite response, CR2, for the simulated data with all the breakdown contours included.

As the dose increases, so does the insensitivity of the breakdown voltage and thus it would be expected that when the sensitivity is considered, the operating point should move towards higher dose values. This is confirmed in figure 7–29, which shows CR2SENS and the limiting value of breakdown voltage. The minimum of the composite response has moved, as predicted, to higher dose values.

Figure 7–30 shows the plots obtained for simulated data using CR2SENSALL as the composite response, ie all response sensitivities considered. For this scenario, the minimum of the composite response is for slightly higher energy values than that of figure 7–29.

In conclusion, the optimum point established without sensitivity analysis is also reasonably insensitive to variations in input factors for all the responses. The

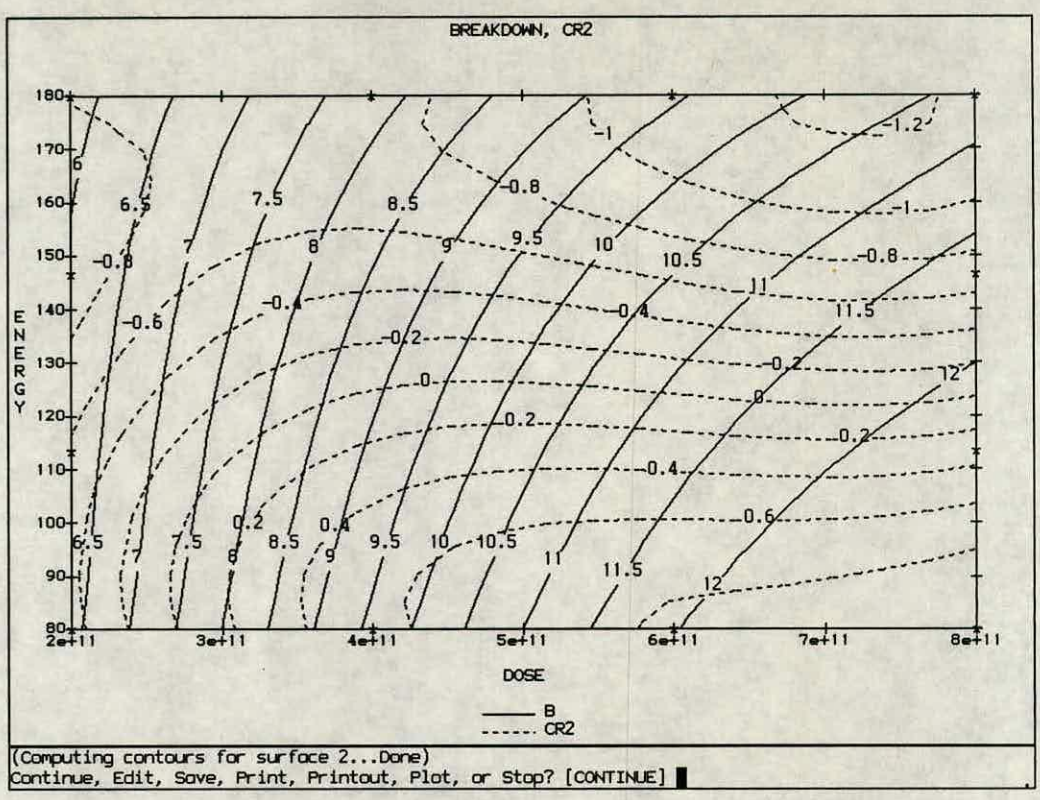


Figure 7–28: Contourplots showing composite response CR2 and breakdown voltage contours for simulated data

final insensitive, optimum set of conditions will still be dependent on the weights given in the composite responses.

Similar results for the real measurement data will be given in chapter 8.

7.6.4 A comparison of Taguchi and RSM

This section deals with the relative merits of using Taguchi and RSM analysis. As outlined in section 7–1, the Taguchi approach is concerned with investigation of the response variation to small noise variation in the input factors. Two values were investigated, namely the mean response and the value of the signal-to-noise ratio.

Figures 7–31 and 7–32 show the results obtained for mean response analysis for variation in dose and energy respectively. The original objectives were defined

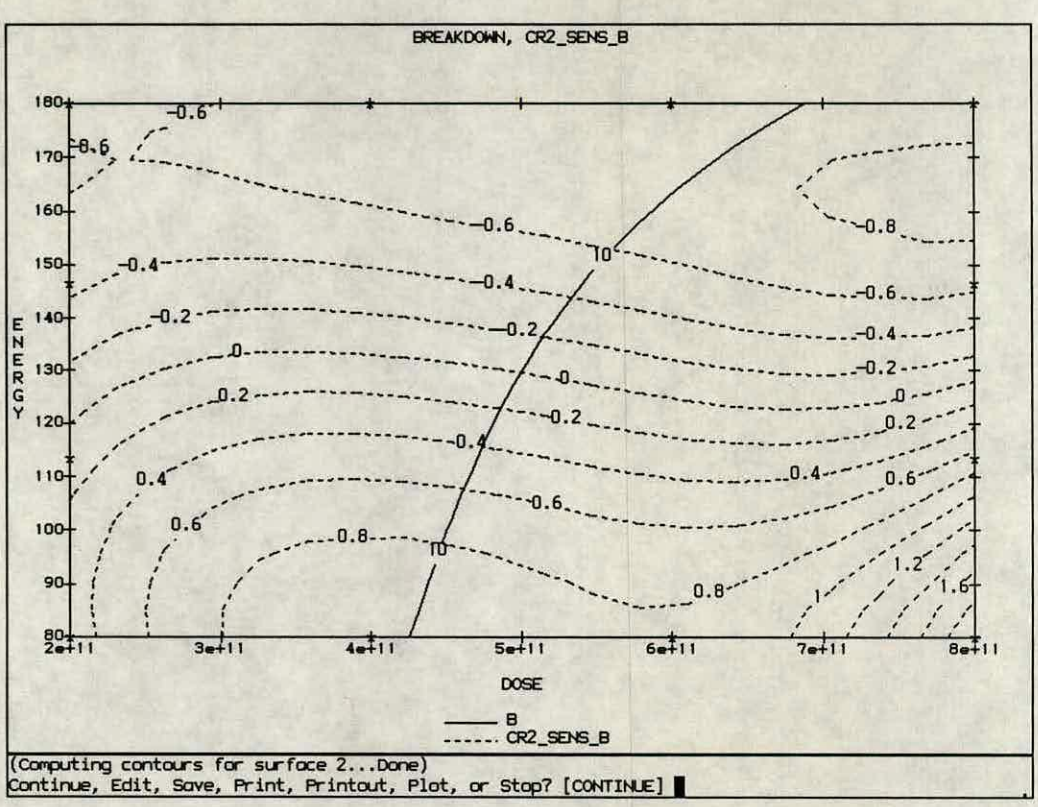


Figure 7–29: Contourplots showing composite response CR2SENS and breakdown voltage contours for simulated data

as breakdown greater than 10V, minimum gamma and subthreshold swing and a small variation in threshold voltage. In order to meet these objectives, firstly for variation in dose, it can be seen from figure 7–31 that for breakdown a high value of dose is required and for gamma and subthreshold swing a low value of dose is necessary. For energy variations, breakdown requires a low value and gamma and subthreshold both indicate a high value of energy is necessary. The results are in complete agreement with those obtained in section 7.6.1 for the RSM analysis.

Figures 7–33 and 7–34 illustrate the results for signal-to-noise ratio analysis. The signal-to-noise ratio has already been described in section 4.2.6. Essentially it is a measure of the variability of a particular response to changes in input factors. The higher the signal-to-noise ratio, the more stable and insensitive the response in question. Thus the objective of the Taguchi analysis is to achieve the desired mean responses with the minimum of sensitivity. From figure 7–33, the most

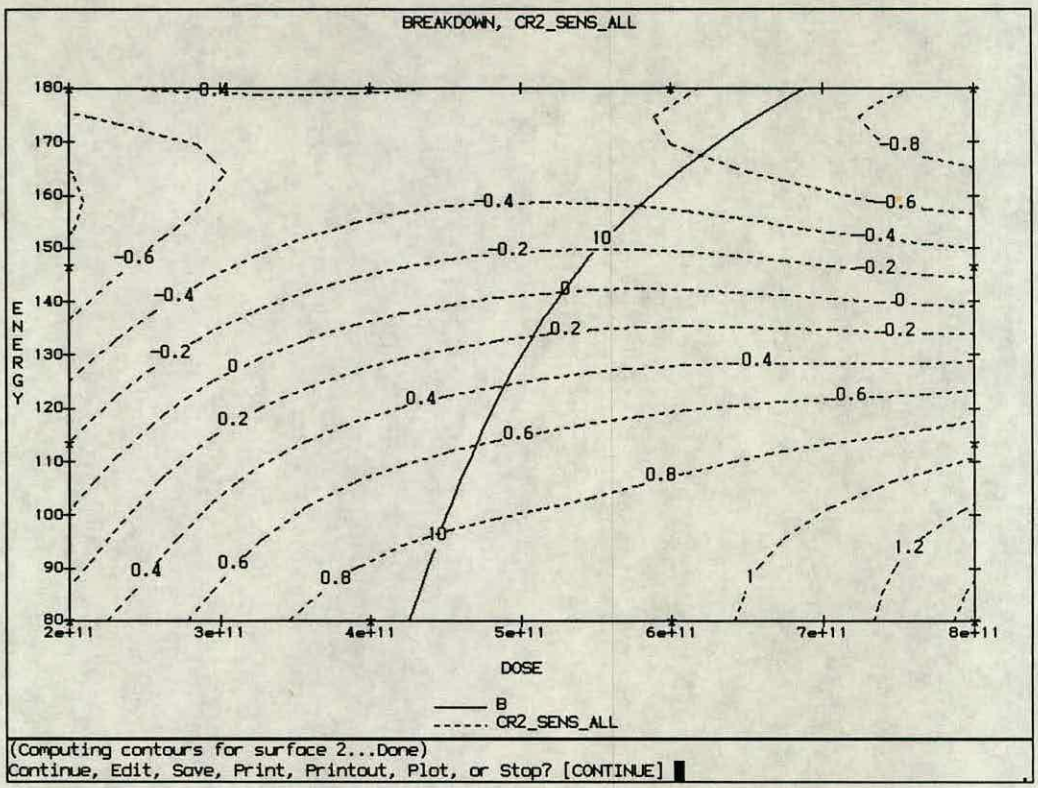


Figure 7–30: Contourplots showing composite response CR2SENSALL and the limiting value of breakdown voltage for simulated data

insensitive areas with respect to variations in dose are high values for breakdown and subthreshold and low values for gamma and threshold voltage. For energy (figure 7–34), the insensitive areas are low values for breakdown, threshold and subthreshold and a high value for gamma.

In order to achieve the optimum set of input conditions, the value of dose must be set at level 2, otherwise the breakdown constraint of 10V will not be met. Since the value of energy does not greatly affect the mean response for breakdown, a level 2 value for energy was chosen since this resulted in minimum gamma and subthreshold swing. A compromise with respect to signal-to-noise ratio was realised using a high value of dose and energy. For breakdown a high dose results in a high signal-to-noise ratio, whereas a high energy leads to a low signal-to-noise ratio, the opposite is the case for gamma. The threshold voltage,

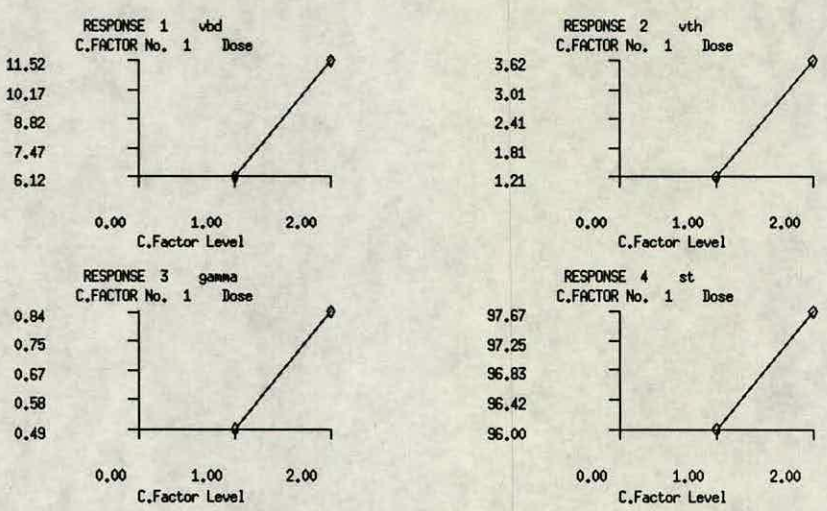


Figure 7–31: The mean values of all the responses for changes in dose

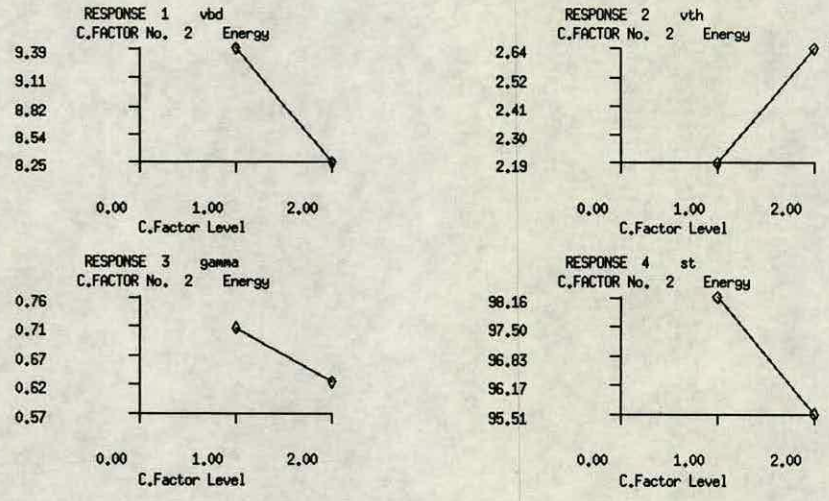


Figure 7–32: The mean values of all the responses for changes in energy

however, is more sensitive at high levels of dose and energy and the subthreshold swing is relatively insensitive, irrespective of the implant levels used.

This analysis is consistent with the results obtained from RSM, ie a high dose and energy. The problem with only using two levels in the Taguchi analysis is that

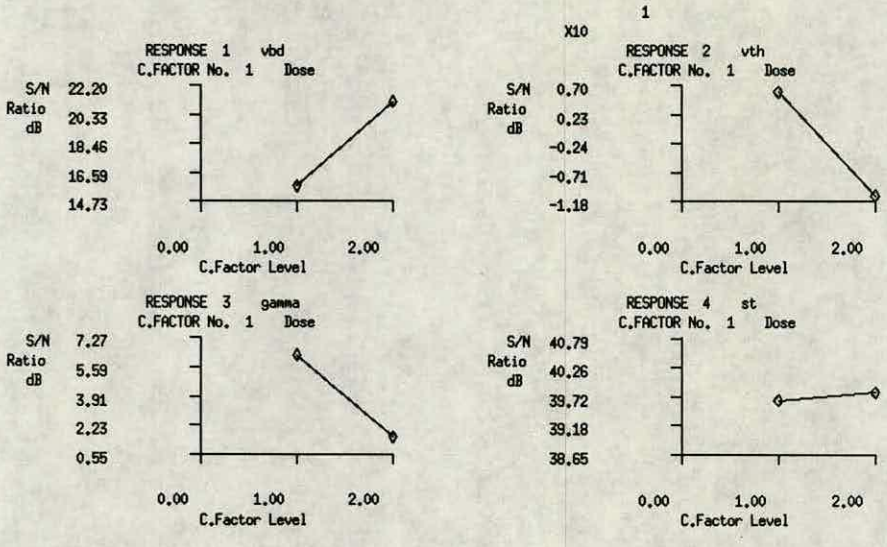


Figure 7–33: The signal-to-noise ratio values for all responses for changes in dose

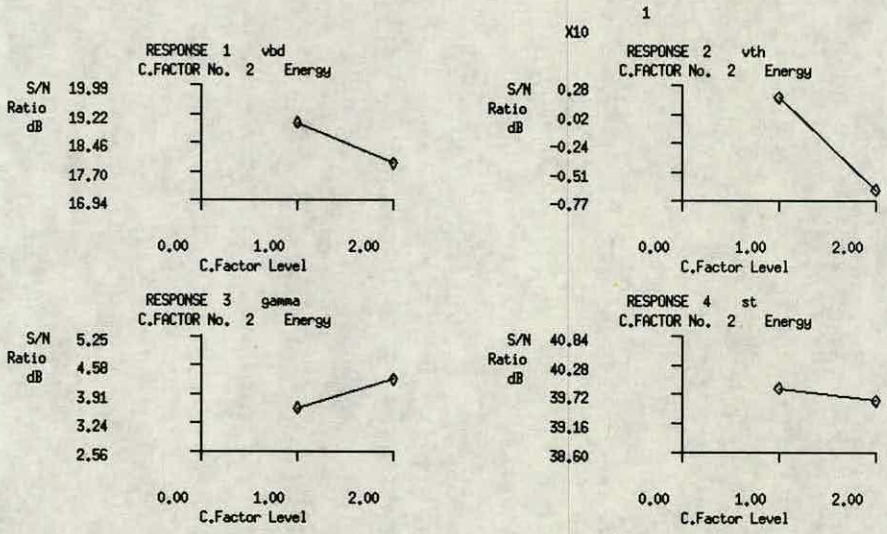


Figure 7–34: The signal-to-noise ratio values for all responses for changes in energy

it is quite possible for the optimum conditions to fall in some intermediate value, which is impossible to determine with only two levels.

The Taguchi approach, however, does offer a structured insight into the main effects and sensitivities of the design and can help in the effort to produce optimised, robust operating conditions.

7.7 Conclusions

This chapter has shown how RSM and simulation can be effectively used in the optimisation of fabrication processes. The structured approach offered by RS/1 means that time and effort is not wasted in unwanted experiments and simulations and that the classical experimental designs result in accurate modelling of the interactions of the various responses.

When used in this manner, simulation can inform the design or development engineer of the major trends resulting from the variation of process parameters. This can greatly reduce the time spent on unnecessary fabrication experiments, thus saving both time and money.

Taguchi analysis is also useful for giving an understanding of the sensitivities of the responses and the main effects of the control factors. The major drawback, however, for Taguchi analysis is that the number of runs can easily become prohibitive, in real life situations. Other disadvantages are that it is difficult to adapt Taguchi to multiparameter optimisation and that it is not conducive to optimisation of values between level settings.

Chapter 8

Test results and analysis

8.1 Introduction

Despite the advances made in the accuracy of models and coefficients for modern simulation programs, it is always important to verify and compare any simulation analysis with real experiments in silicon. To this end, for each of the experimental points suggested by RS/1 and simulated in the previous chapter, two wafers were fabricated here in the clean room at the Edinburgh Microfabrication Facility. The fact that two wafers were fabricated for each run, allowed for any wafer breakages and also compensated for any wafer to wafer variations.

This chapter describes the basic steps in the 1.5 μm nMOS process and outlines some fabrication techniques that were employed to ensure the reliability and accuracy of the results. Included are the results obtained from measurement of the wafers and the corresponding plots from RS/1. For comparison purposes, the RS/1 plots from the simulation results for both D-optimal and full-factorial are also outlined.

The optimum operating point and least sensitive areas are outlined and a comparison is made between simulation and reality in this respect.

8.2 The 1.5μm nMOS process

The following is a brief list of the main steps involved in the 1.5 μm nMOS process.

1. Define active regions.
2. Perform field implant (boron).
3. Grow field oxide.
4. Perform depletion implant (arsenic)
5. Grow gate oxide.
6. Perform double implant (boron). Threshold and breakdown adjust.
7. Polysilicon deposition.
8. Gate length photolithography step.
9. Polysilicon etch.
10. Source/drain implant.
11. Reflow operations.
12. Contact photolithography step.
13. Oxide etch to clear contacts.
14. Aluminium deposition, photolithography, RIE and wet etch.

8.3 Wafer processing

This section describes the different techniques used during processing to ensure reliable and accurate results. The main areas where attention was focussed were:

- Step 8. Gate length photolithography step.
- Steps 13 and 14. Formation of contact holes.
- Step 14. Aluminium etch.

Gate length photolithography step As the name suggests, this lithography step was responsible for defining the length of the gate. Much care was required at this stage to ensure that the right focus and exposure was used on the stepper to produce the correct geometries on the wafer. This was essential to ensure that the simulated geometry and wafer geometry were the same.

In order to guarantee this, a focus/exposure matrix was generated across a test wafer. Figure 8–1 shows the method and values used in the matrix. The focus was varied from 0 to 6 and the exposure from 900 ms to 1400 ms.

After exposure and development, the wafer was measured using a KLA 5015 [114] to establish the gate length. This allowed for any changes in geometry after etching. On analysis of these results, it was decided to print the rest of the wafers using a fixed focus of 2 and an exposure matrix of 900 up to 1400 ms across the wafer, thus ensuring exact geometries in one column of the wafer.

At the end of the fabrication process, further analysis showed the 1300 ms column to represent the correct geometries and as a result this column was used in the testing of the transistors.

Formation of contact holes Attention was focussed at this part of the process to ensure that (a) the correct exposure was used to open the contact holes and (b)

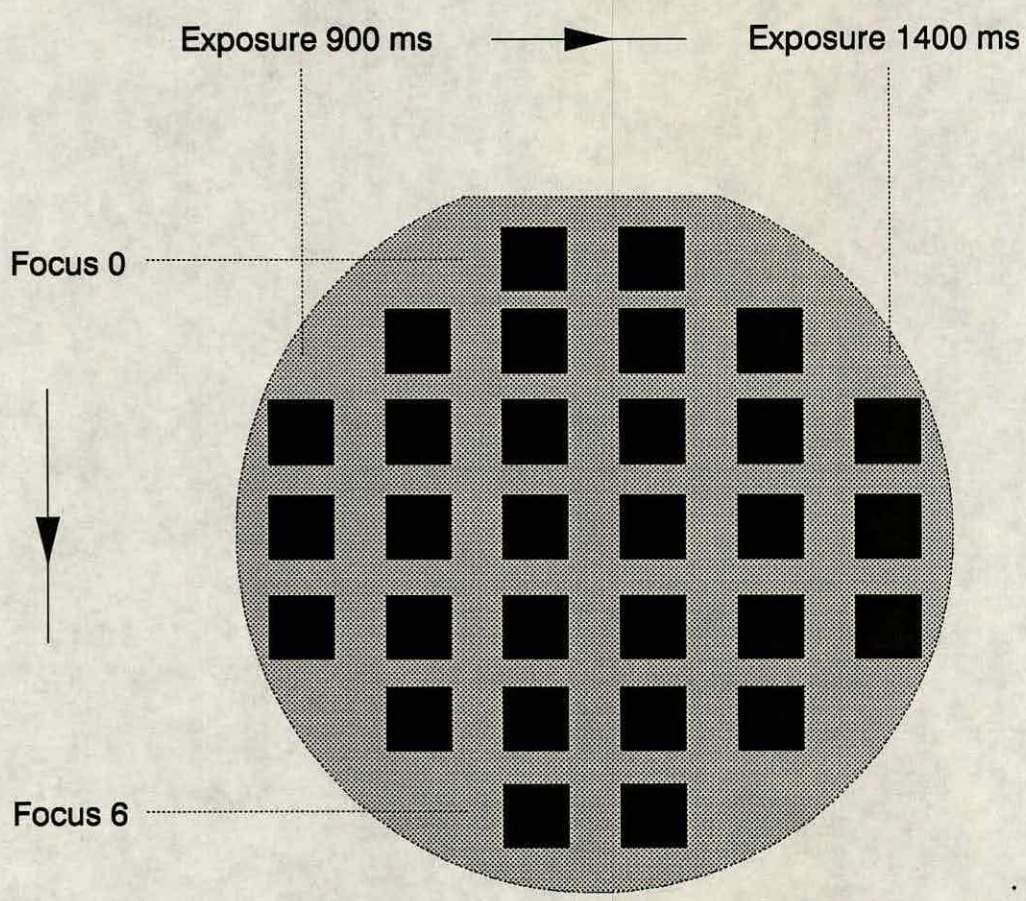


Figure 8–1: Focus/exposure matrix used to establish correct wafer geometries

all the oxide was removed from the surface to guarantee good electrical contact. As for the previous section, a test wafer was used to establish the exposure at 2000 ms as adequate for opening of the contact holes. If the exposure time is too small then the contact holes will not be opened properly or not at all. After oxide etch, each wafer was tested in 5 areas using a Nanospec to check that no oxide remained on the surface before the deposition of aluminium.

Aluminium etch This stage of the process was considered crucial for the operation of the final devices. The first part of the etch was performed using reactive ion etch (RIE) techniques and the second stage using a wet etch in phosphoric acid.

Experiments were performed using the RIE machine to establish the optimum operating conditions for this particular process. The parameters varied were: etch

time, etch power and base plate temperature. These were eventually optimised at 8 minutes, 500 W and 37 deg C respectively. Since RIE etching is anisotropic ie only in one direction, the vertical, it was also necessary to use a short wet etch or "wet dip" to remove any excess aluminium in the form of fillets. There is a trade off between the RIE etch and the wet etch. Too long an RIE etch can lead to damage of the photoresist, albeit removal of most of the aluminium. Too short an RIE etch means that much aluminium still remains and a longer wet etch is required. Since wet etching is isotropic ie in all directions, then if the wet etch time is too excessive the aluminium can be undercut and lead to the breaking of tracks. Great care was taken at this stage to optimise both the RIE and wet etch processes.

After completion of the fabrication process, the wafers were then tested for correct operation. This will be outlined in the following section, together with a comparison with simulation.

8.4 Test results

The wafers were tested using a standard probe card and an HP 4145B parameter analyser. Averages were taken for all measurements on different die or wafers with the same dose and implant conditions. Table 8–1 shows the results obtained for the wafer measurements.

Figure 8–2 shows the comparison of a set of gate characteristic curves for reality and simulation. Both curves are for implant conditions of 7×10^{11} atoms cm $^{-2}$ and 140 keV.

In order to illustrate the effect of exposure time (for step 8 in section 8.2) on the breakdown voltage, measurements were taken across the wafer for the different exposures. The smaller the exposure time the larger the channel length, hence a larger breakdown voltage and vice versa. Figure 8–3 shows how the breakdown voltage varies with exposure time for three different nominal channel lengths,

Dose /atoms cm^{-2}	Energy /keV	V_{bd}/V	$\gamma/V^{\frac{1}{2}}$	S_t / mv/dec	V_t/V
2e11	80	6.0	0.478	92.3	0.56
4e11	80	8.8	0.630	97.5	0.66
6e11	80	11	0.866	97.9	0.70
8e11	80	12.30	0.912	99.5	0.76
2e11	113	6.0	0.538	91.3	0.53
4e11	113	10.3	0.747	96.4	0.625
6e11	113	10.5	0.805	96.5	0.64
8e11	113	12.5	0.888	97.4	0.70
2e11	147	6.0	0.602	89.5	0.52
4e11	147	10	0.679	93.5	0.58
6e11	147	10.4	0.733	96.5	0.61
8e11	147	11.5	0.767	95.86	0.60
2e11	180	6.0	0.497	88.9	0.49
4e11	180	8.8	0.699	91.2	0.54
6e11	180	10.6	0.722	91.6	0.55
8e11	180	11.5	0.814	91.6	0.55

Table 8–1: Results for wafer measurements

namely 0.8, 1 and 1.5 μm. The wafer measured had a dose of 2×10^{11} and an energy of 147 keV.

8.5 Analysis of results

8.5.1 RSM analysis

This section outlines the results obtained for the real measurement data. The results were loaded into RS/1, a model was fitted and analysis performed. The values for R-squared error are given in table 8–2. As was the case for simulation, all values for the R-squared error were greater than 0.95, indicating a good fit.

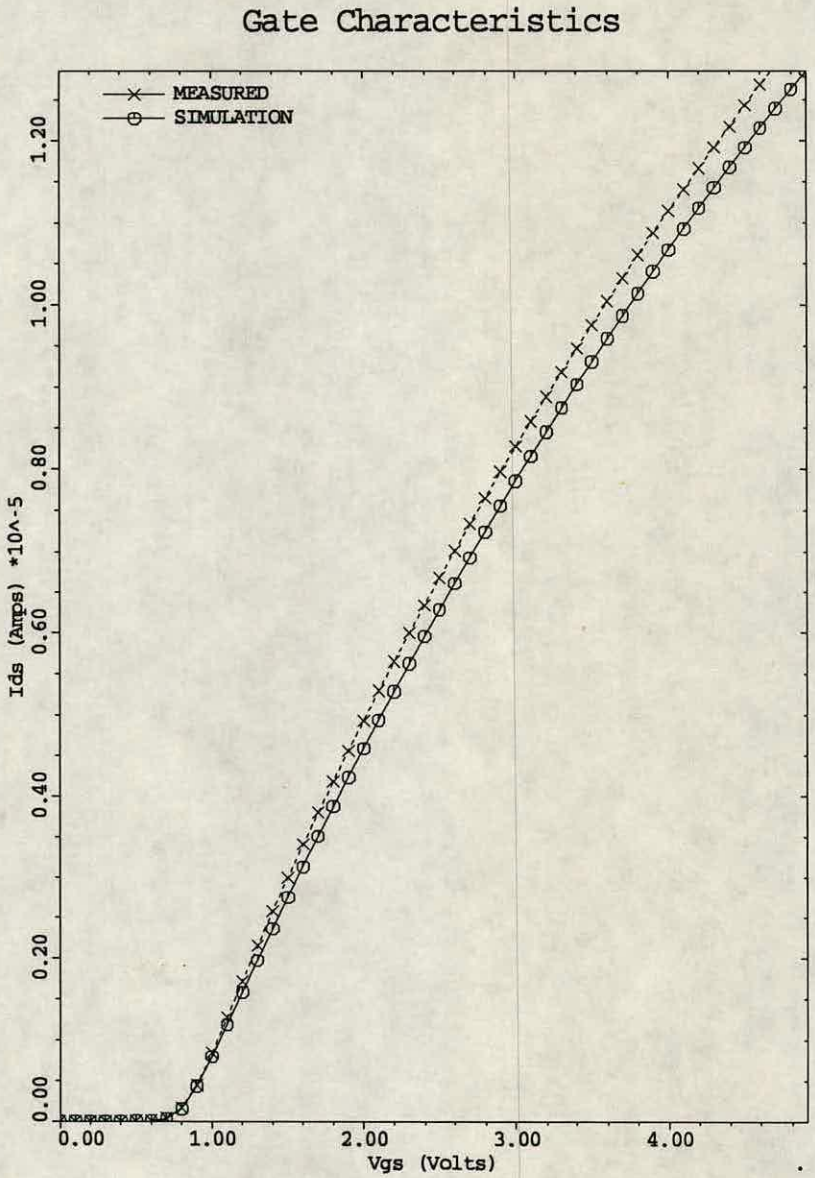


Figure 8–2: A comparison of a set of IV curves for simulation and reality

Response	R-sq error real
breakdown	0.9984*
gamma	0.9589
subthreshold	0.9926
threshold	0.9922

Table 8–2: R-squared errors for the fitting of models for real devices

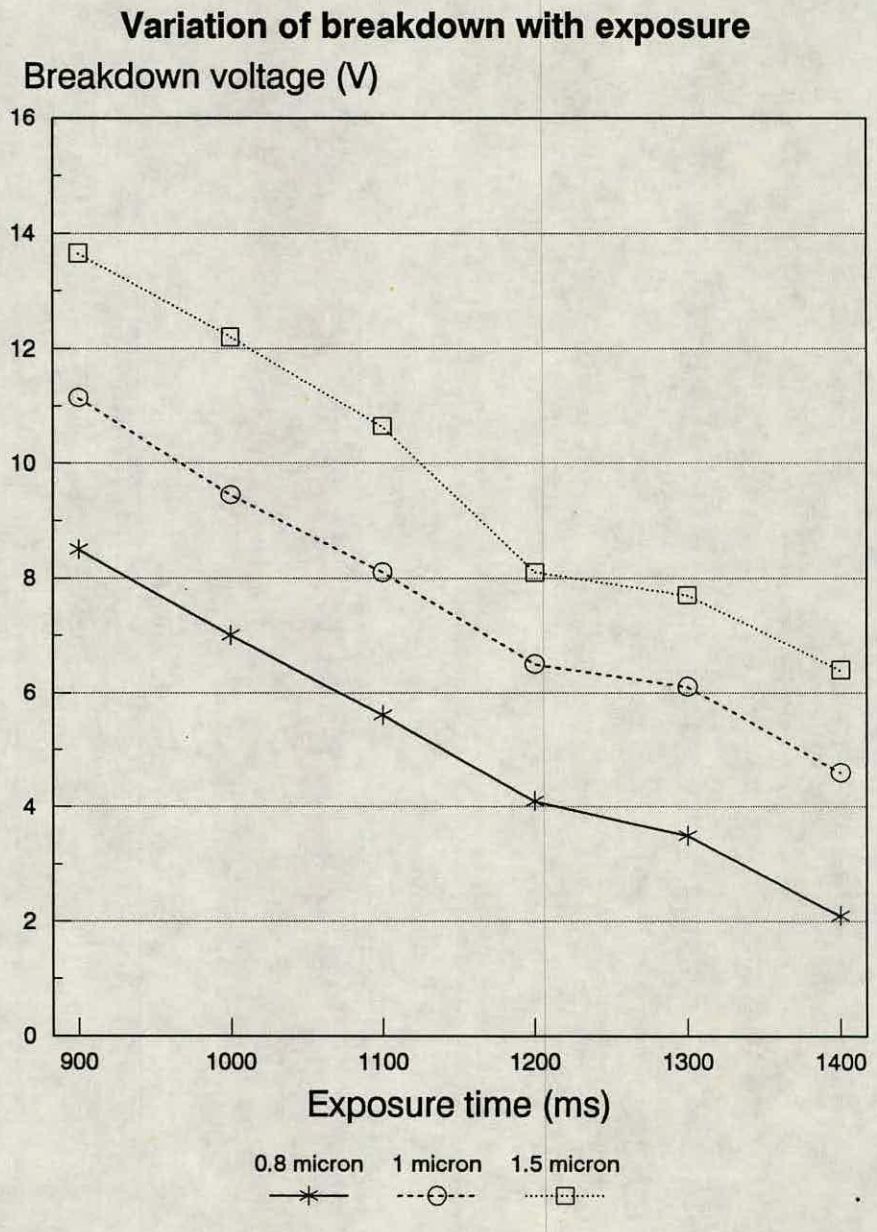


Figure 8–3: Variation of breakdown voltage with exposure time for different nominal channel lengths

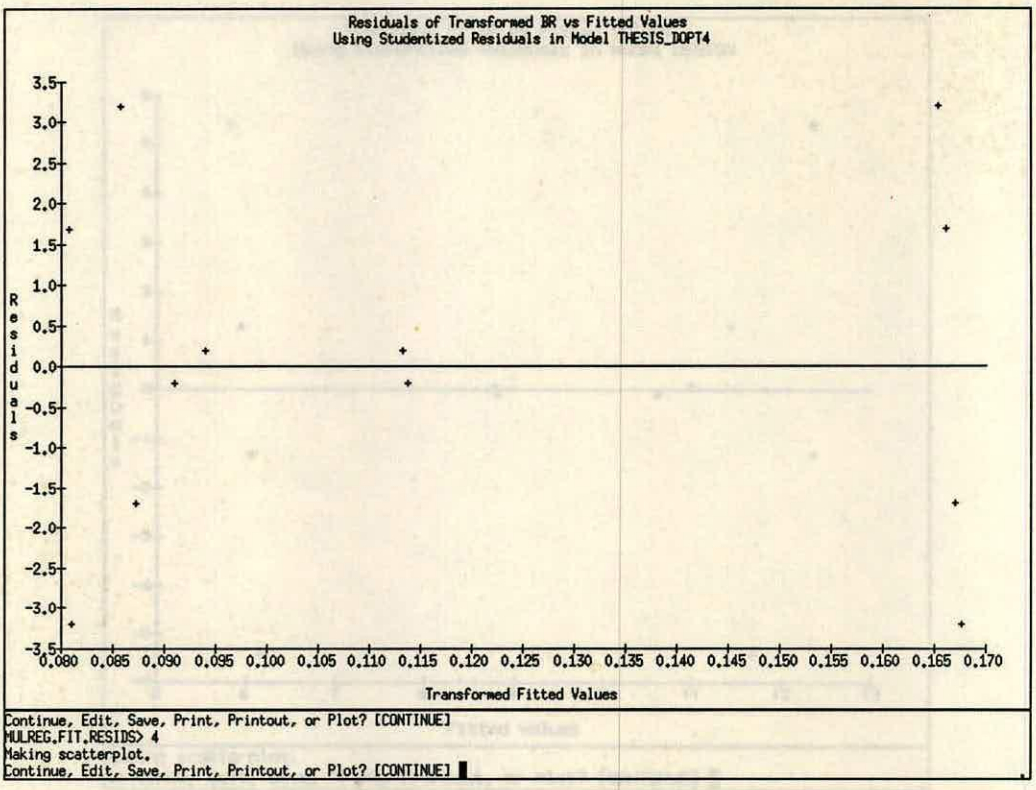


Figure 8–4: Scatterplot of real breakdown residuals

The '*' indicates that the response has been transformed, in this case using a reciprocal transformation.

The residuals table for each response was analysed to ensure that the variation in the residuals was random in nature, resulting in an adequate model fit. Figure 8–4 shows the scatterplot obtained for breakdown voltage. The residuals are similar to those obtained for the simulated data. Similar plots were also generated for gamma, subthreshold swing and threshold voltage.

Contourplots

This section outlines the different contourplots obtained from RS/1. Figures 8–5, 8–6, 8–7 and 8–8 describe the results for real devices obtained for the D-optimal experiment for breakdown, gamma, subthreshold swing and threshold voltage respectively. Also shown on these plots, for comparison purposes, are the contourplots obtained for simulation.

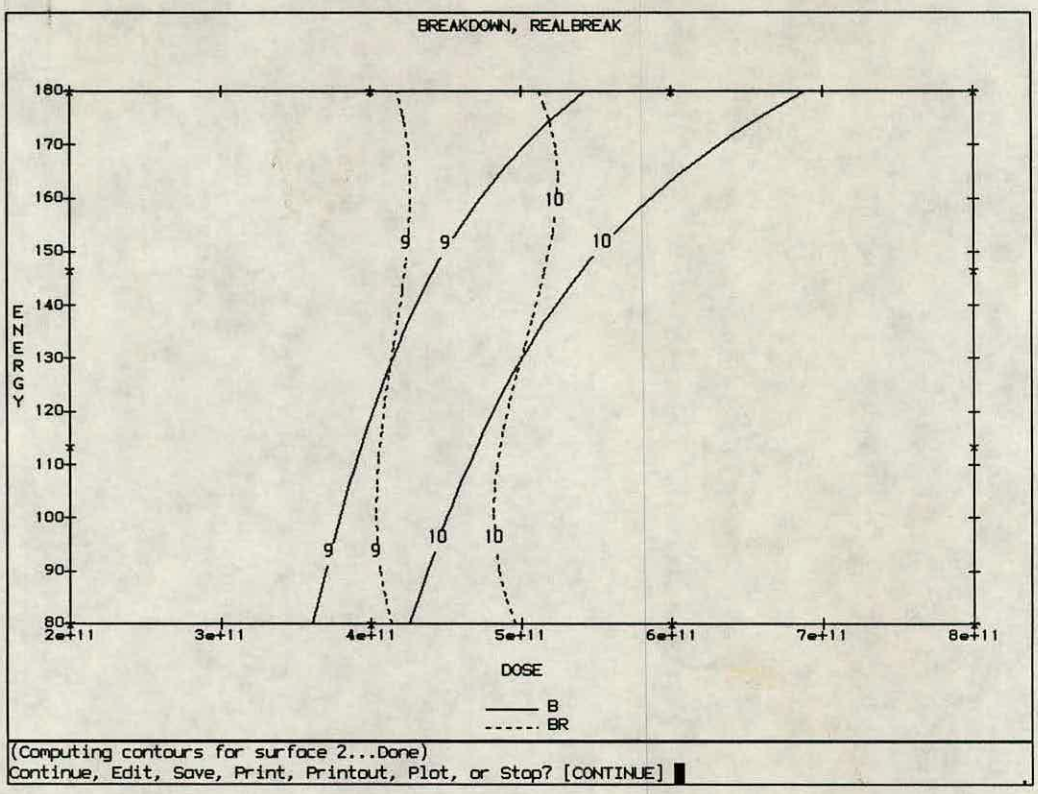


Figure 8–5: A contourplot of simulated and real data for breakdown analysis. B refers to the simulated results and BR the real results

Breakdown was divided into three areas, less than 9V, 9V - 10V and greater than 10V. Less than 9V was considered unsatisfactory, 9V - 10V marginal, and greater than 10V satisfactory. These areas are detailed in figure 8–5.

As can be seen from the figures, the simulated data and the real data show good agreement with each other. Any discrepancies can be easily explained by slight inaccuracies in the simulation and RS/1 models used, together with experimental error or variations.

8.5.2 Optimum operating conditions

This section outlines the different contourplots obtained in the establishment of the optimum operating conditions. Due to the inadequacies of the response, CR1, discussed previously in section 7.6.2, the second definition of composite response, CR2, was used for the following analysis.

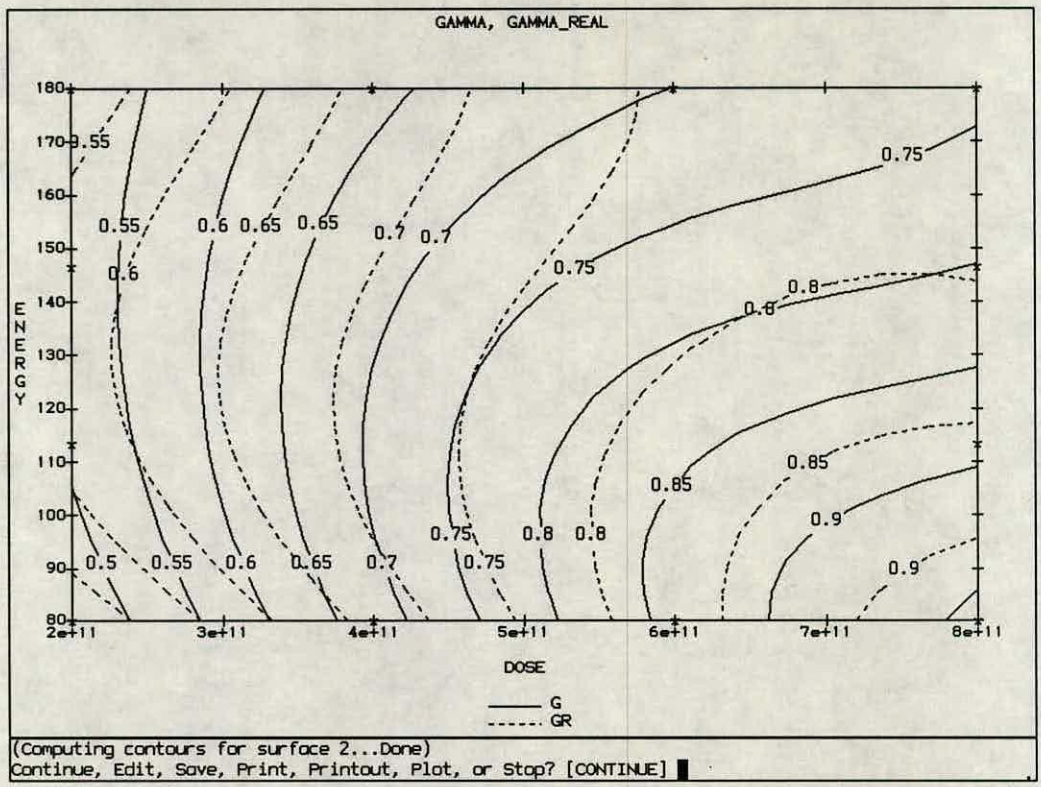


Figure 8–6: A contourplot of simulated and real data for breakdown analysis. G refers to the simulated results and GR the real results

Figure 8–9 shows the measured results obtained for CR2 and the limiting values of breakdown and threshold voltage.

The optimum operating point was extracted from RS/1, given the constraints of breakdown and threshold voltage. The value of dose was 6×10^{11} and an energy of 178 keV. This shows good agreement, not only between simulation and reality, but also on a qualitative analysis of the actual data.

As for the simulation case, investigation was performed to establish the effects of removing breakdown voltage from the composite response expression. Figure 8–10 shows the contourplots for CR2 without a term for breakdown.

The plot is similar to that for figure 8–9, which includes breakdown in the expression. In addition the optimum operating point is in approximately the same area of parameter space.

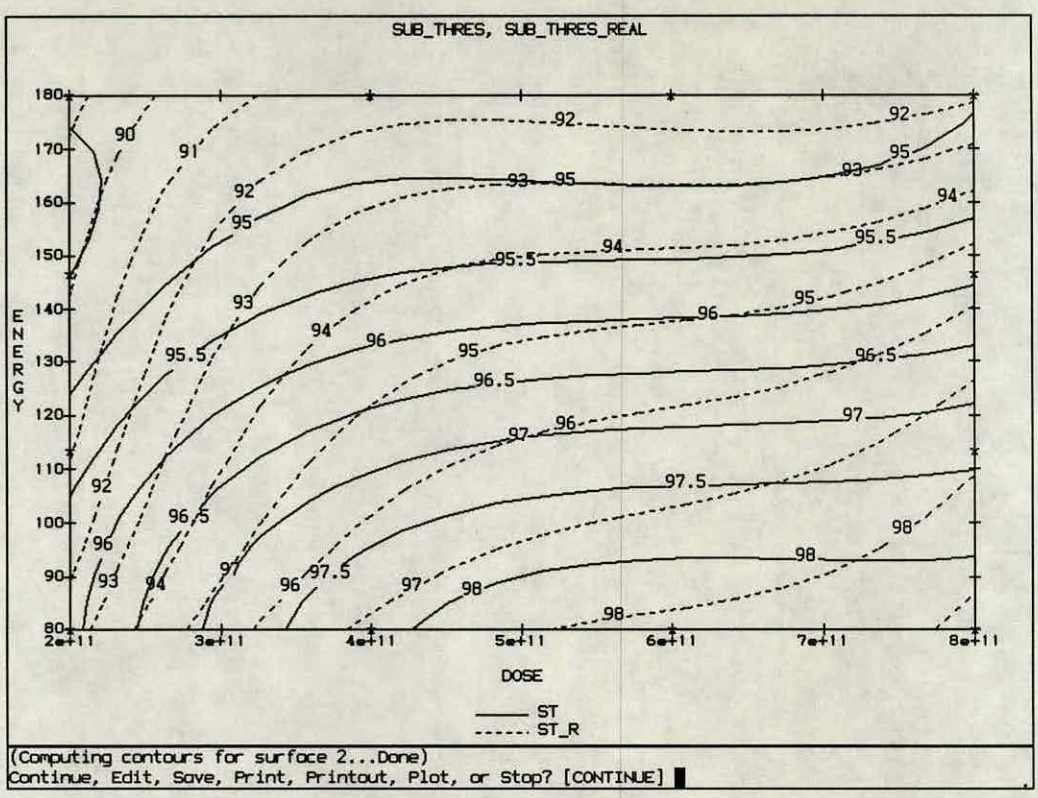


Figure 8–7: A contourplot of simulated and real data for breakdown analysis. ST refers to the simulated results and ST_R the real results

8.5.3 Sensitivity analysis

This section addresses the issues involved in reducing the sensitivities of the device characteristics for the real devices.

Figure 8–11 shows the plots obtained for the measured data using CR2SENSALL as the composite response, ie all response sensitivities considered. For this scenario, the minimum of the composite response is for similar implant conditions to that of figure 8–9, indicating that the original optimum operating point is also insensitive to manufacturing variations.

Also shown in figure 8–11 is the existing operating point, P. This shows the improvement to be gained using the new optimised operating point, O.

In conclusion it has been shown that the simulated and measured data show good agreement with respect to the optimised and insensitive design.

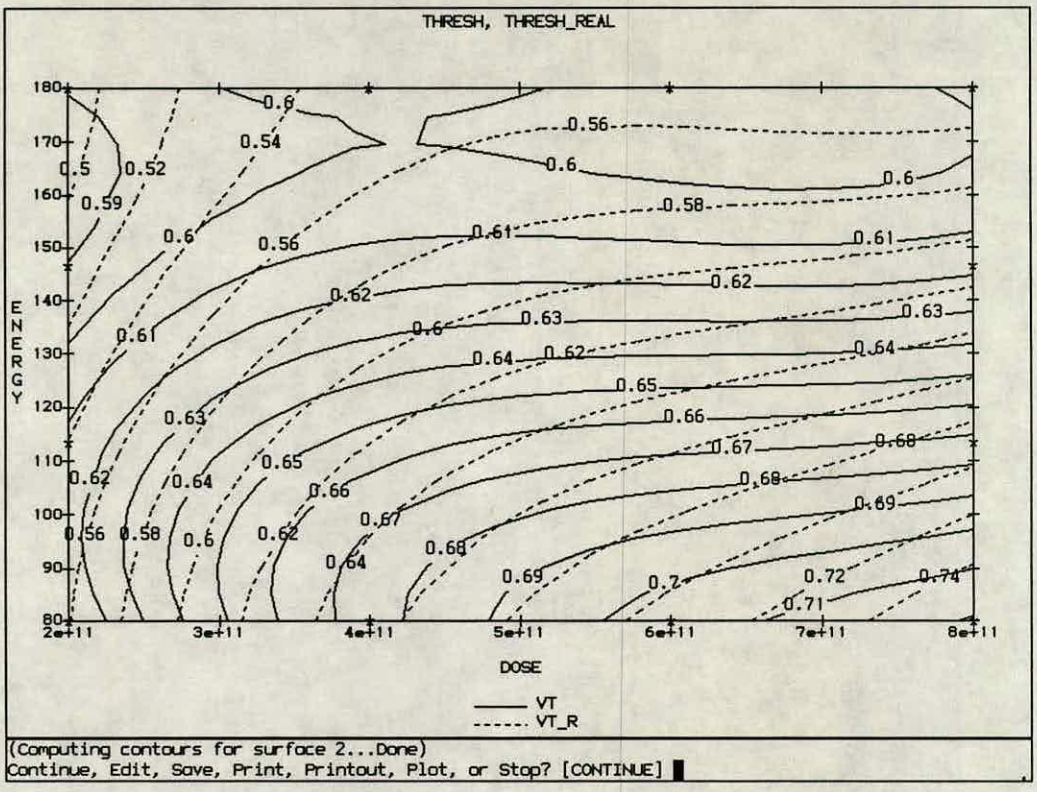


Figure 8–8: A contourplot of simulated and real data for breakdown analysis. VT refers to the simulated results and VT_R the real results

8.5.4 Comparison of D-optimal and full-factorial

This brief section compares the results obtained from RS/1 for D-optimal and full-factorial analysis. The former required 12 runs, whereas the latter needed 16 runs.

Comparisons will be made by considering the plots obtained when the sensitivities for all the responses were included for both simulation and reality. These plots have already been illustrated for the D-optimal design in figure 7–30 and figure 8–11. These show the results for simulation and reality respectively.

Figures 8–12 and 8–13 show the equivalent plots for the full-factorial design.

A brief study of these four figures shows a good agreement between the D-optimal and full-factorial designs. All indicate that an average to high value of dose and an energy of about 180 keV are required for optimum performance. This

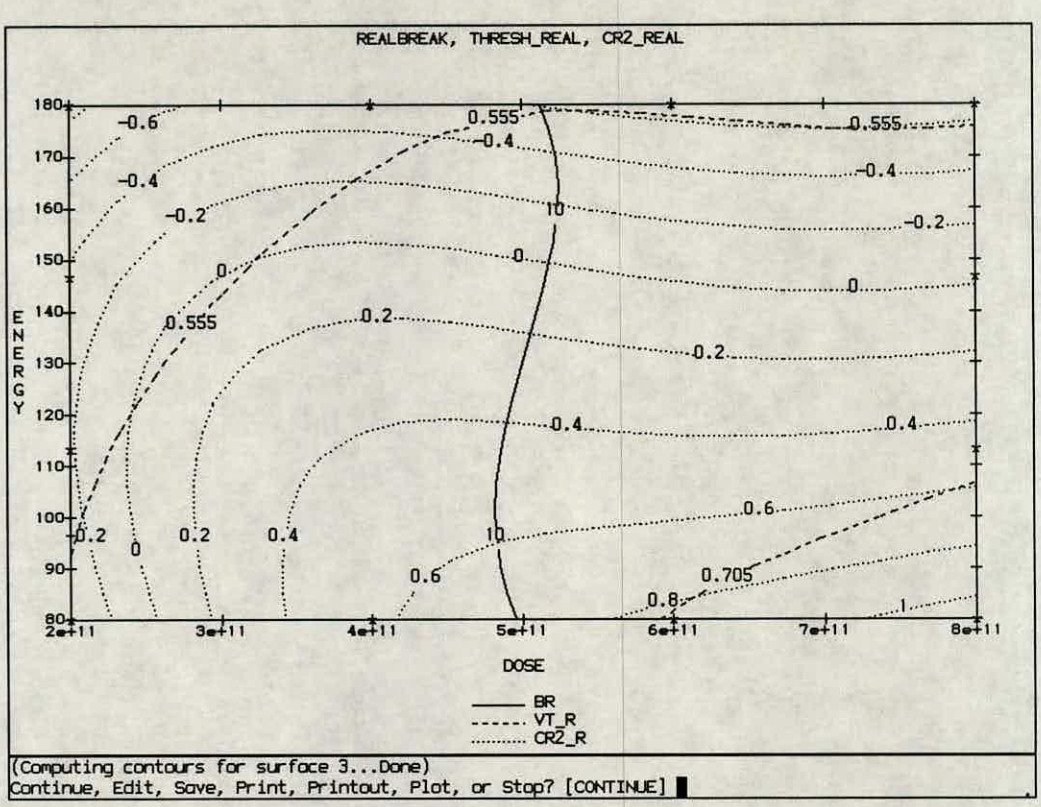


Figure 8–9: Contourplots for real data, defining the optimum region for the second definition of composite response

shows the savings that are possible using these statistical designs instead of a full-factorial analysis. Although in this situation only four experimental runs were saved, for larger numbers of input factors the savings would be even greater.

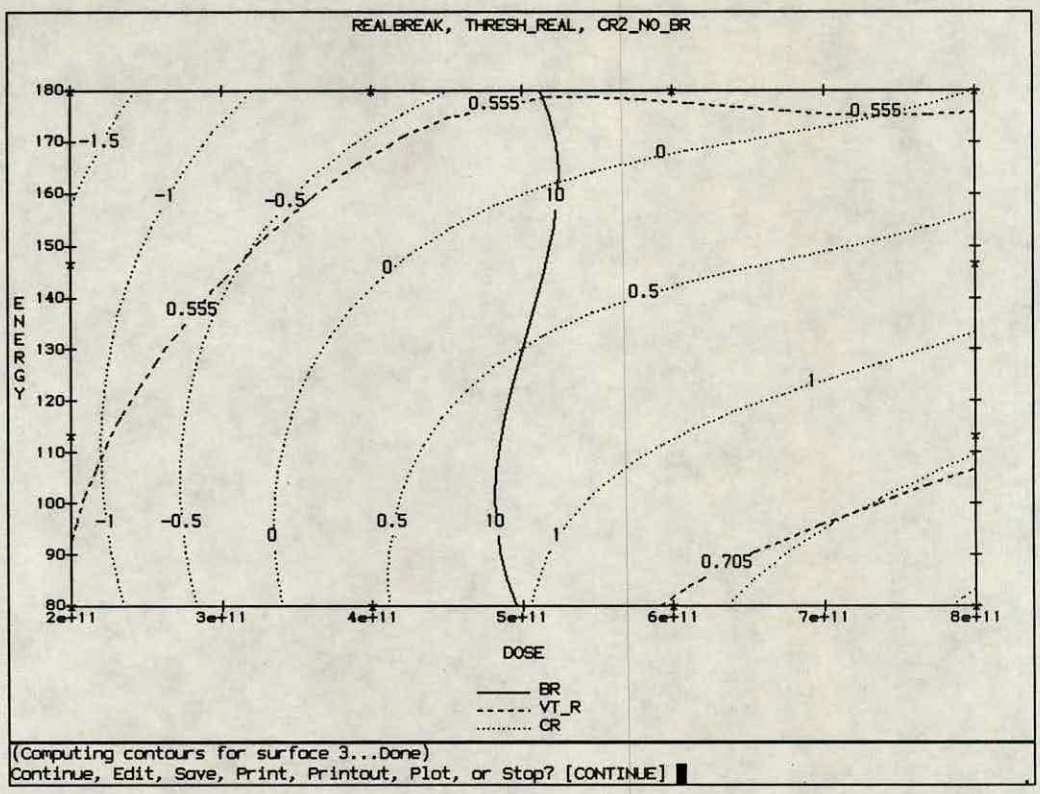


Figure 8–10: Contourplots for simulated data, defining the optimum region for the second definition of composite response without a term for breakdown voltage

8.6 Conclusions

This chapter has shown how statistical techniques can be used in the optimisation of semiconductor processes, both for simulation analysis and also in a real life situation. Using powerful software packages such as RS/1, composite responses can be generated which can be used to both establish an initial optimum operating point and then to arrive at a robust, insensitive design.

The statistical designs available in RS/1 can reduce the number of experimental runs required when compared to full-factorial designs. A comparison of a D-optimal design and a full-factorial design show good agreement for both simulation and reality.

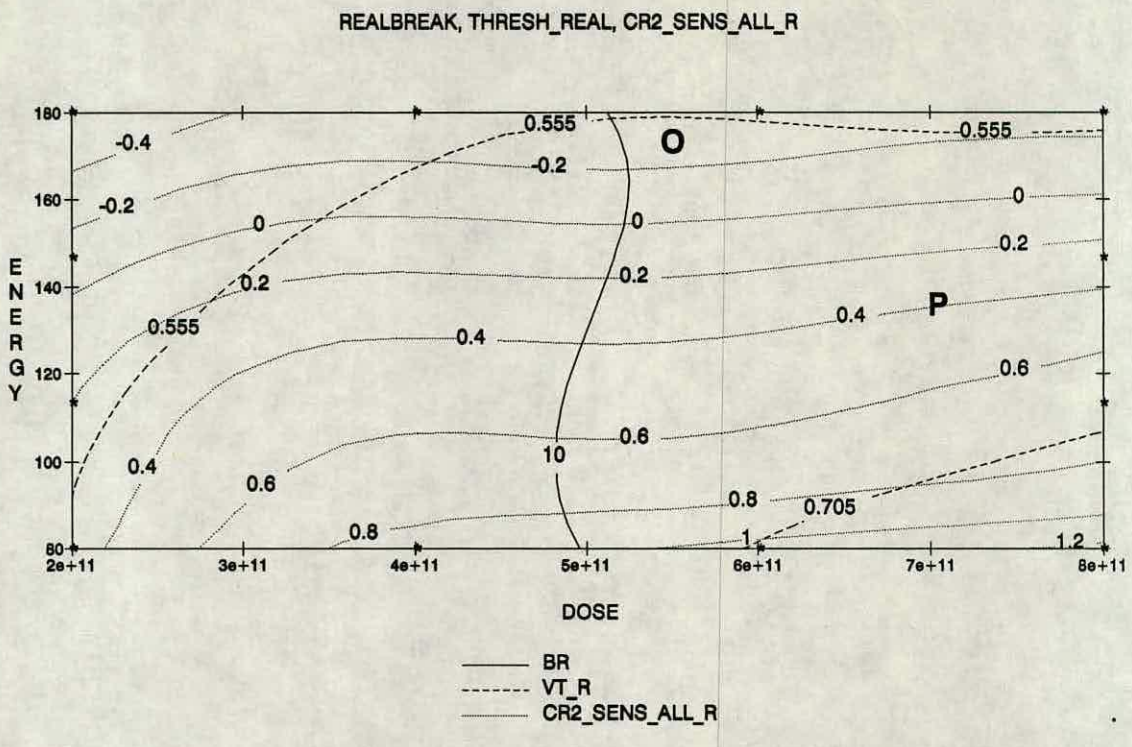


Figure 8–11: Contourplots showing composite response CR2SENSALL and the limiting value of breakdown voltage for real data

A new operating point for the double implant of the EMF 1.5μm nMOS process was established. A comparison with the old operating conditions shows the benefits that can be gained from this type of approach.

The verification of the simulation results with the real devices illustrates with increased clarity the usefulness of the simulation approach to solve present manufacturing problems or tradeoffs in the semiconductor industry.

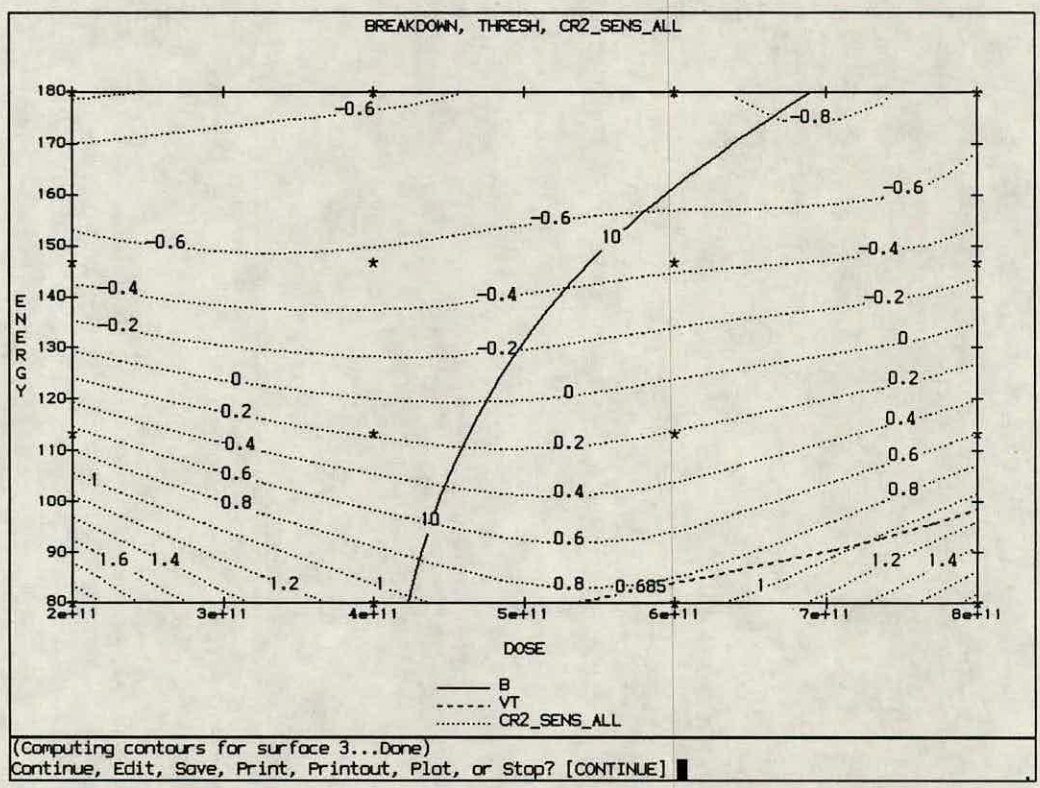


Figure 8–12: Contourplots for composite response CR2SENSALL for simulated data and a full-factorial design

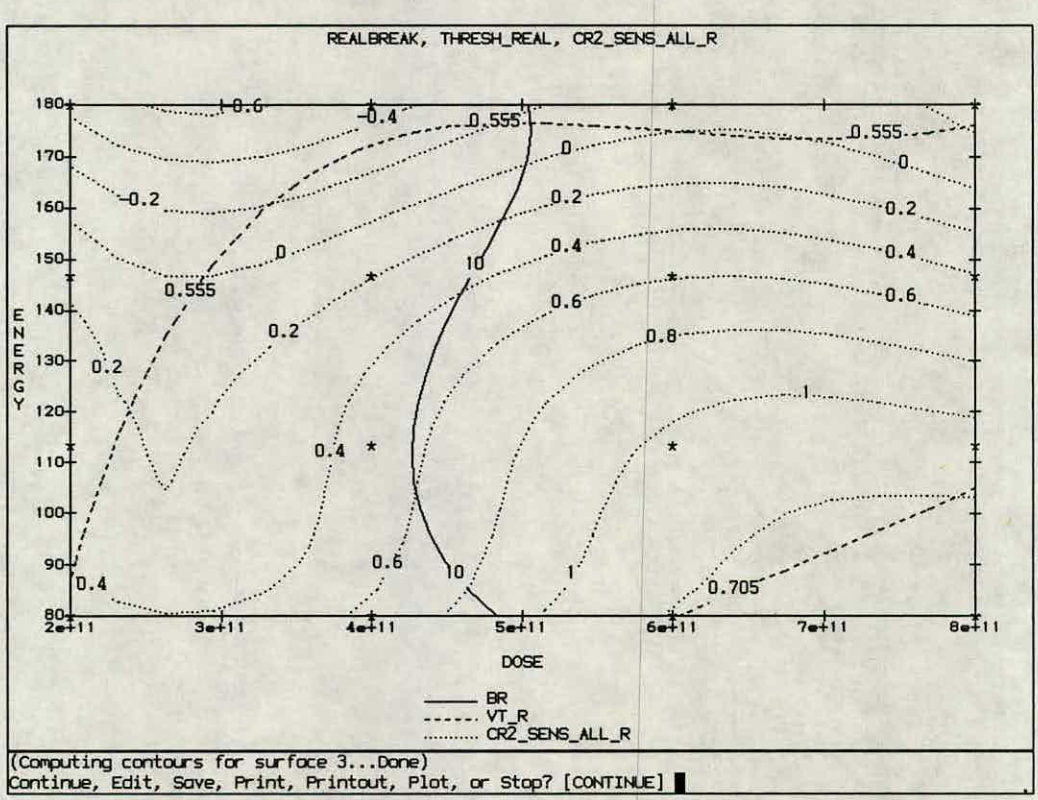


Figure 8–13: Contourplots for composite response CR2SENSALL for real data and a full-factorial design

Chapter 9

Conclusions

Efficient process design and tighter process control often offers a competitive edge to today's complex and aggressive VLSI manufacturing environment. The issue has been further complicated with the strain placed on semiconductor companies to produce devices with smaller geometries and higher yield, while reducing the time to market. No longer can quality be an afterthought in the manufacturing process, but it must be part of that very process.

It was the realisation of this that led to Design for Manufacturability (DFM) techniques being implemented in many manufacturing companies, not just restricted to the semiconductor industry. This is a structured, rigorous methodology where the main question asked is "Is the product manufacturable with a sufficient yield".

The issues involved in semiconductor manufacturing are complex, with trade-offs prevalent for different device characteristics. It is thus essential to use techniques such as design of experiments and Response Surface Methodology to gain greater understanding of the complex interactions. An experimental approach to this is often too expensive to be feasible and too slow, given the competitive environment. This has led to the use of simulation programs to reduce the cost and time of experimentation using statistical techniques.

The main thrust of this work has been to show how a software simulation system can be used, along with the design of experiments technique, to produce

optimised, robust and insensitive designs. A brief outline of the major results and conclusions is presented here, together with some ideas for future work.

In **chapter 2** it was shown how process simulation packages can be used as an aid in the semiconductor industry. It was outlined how the product development cycle can be reduced by use of these tools and how the complex interactions in integrated circuits can be better understood.

Chapter 3 showed how process simulation can be linked with device simulation and parameter extraction packages to gain further insight into device operation and to extract circuit parameters. These parameters can then be utilised to characterise the device and used in circuit simulation packages.

Although the utilisation of CAD simulation packages can greatly reduce the time to market for a new product, if linked with design of experiments and statistical techniques the development time can be reduced even further. **Chapter 4** gave an overview of some of the different methods involved in design of experiments and statistical analysis. In particular it described how Response Surface Methodology and Taguchi analysis can be used in the optimisation of semiconductor processes. It also illustrated how the statistical package RS/1 can be used as a tool for these optimisation methods.

Despite the availability of increased processing power, many hours or days can elapse before the results of simulations can be obtained. This has resulted in the use of parallel processing techniques to reduce the simulation time. In **chapter 5** the evolution of parallel computing and its present status was outlined. It was shown how it can be used to perform computationally intensive tasks that would otherwise be impossible to implement.

Chapter 6 described the operation of an automated software system using commercial simulation software, the RS/1 package and parallel processing. Three main types of hardware platforms were utilised, namely a transputer taskfarm, a taskfarm using Intel i860s and a SUN4 workstation network.

It was shown that the porting of commercial simulation software onto different platforms was not a trivial task. The problem was exacerbated due to non standard

software and compilers. It was found that code with Unix specific system calls required much modification to facilitate its operation, especially on the transputer systems. In general Fortran, due to its standardisation, was more portable across the different platforms.

The transputer taskfarm system was found to be extremely flexible in its application to any user source code. It produced a significant speedup when the time taken to run 10 jobs on 10 transputers was compared to 10 sequential simulations on a SUN4 workstation. It was ported onto two different transputer platforms, a PC rack system and a Meiko Computing Surface. The Meiko taskfarm was found to give superior results when compared to the PC taskfarm, due to more efficient hardware.

The parallel taskfarm using Intel i860 processors produced the most speedup when all systems were compared. This was due to the high processing capability of the i860 and the dedicated architecture of the Computing Surface.

The SUN4 network also resulted in a significant speedup when compared to the sequential SUN4 operation. It was found that although not as fast as the Intel taskfarm, it produced more favourable results than the transputer taskfarm.

The optimum hardware configuration for this type of computation must ultimately depend on the user. The least expensive system is the PC transputer taskfarm and although it results in a significant speedup, it is not as powerful as either the Intel system or the SUN4 network. The Intel taskfarm is the most powerful of the three systems but has the disadvantage over the SUN4 as being less flexible. As a result the final choice must lie with the end user after cost, efficiency and flexibility considerations are taken into account.

Given the flexibility of the SUN4 network, an interesting continuation to this work would be the development of a graphical user interface system to control the taskfarm and to include the automation software for generation and extraction of the simulation files. The ease of use of a menu driven system would greatly contribute to the overall user-friendliness of the system, with the parallel taskfarm being transparent to the user.

In order to prove the theory of IC optimisation, an example was taken, using the EMF 1.5μm nMOS process. **Chapter 7** showed how design of experiments and simulation can be effectively used in a practical example for the optimisation of fabrication processes. The part of the process optimised was a breakdown adjust implant. It was found that using a composite response function it was possible to arrive at the optimum conditions for the different control factors, with respect to the various responses. This resulted in a considerable improvement over the existing implant conditions.

Using partial derivatives it was possible to perform sensitivity analysis in order to establish the most insensitive operating region. This was considered essential in developing and finally producing a manufacturable device.

The structured approach offered by RS/1 means that time and effort is not wasted in unwanted experiments and simulations, thus reducing product development times even further.

A comparison of RSM and Taguchi results showed good agreement. Both techniques indicated a high dose and a high energy as optimum processing conditions, although only two factor levels and two noise levels were used for the Taguchi analysis. If three factor levels and three noise factor levels had been used this would have required 81 runs. This would not be practical for most situations.

Although the Taguchi technique was prohibitive with regard to the number of experimental runs, it was useful for giving an understanding of the sensitivities of the responses and the main effects of the control factors. Other disadvantages of Taguchi analysis were that it was more difficult to adapt it to multiparameter optimisation and it was not conducive to optimisation of parameter values between level settings.

The benefits of the RSM analysis were that much information could be gained from a reduced number of runs and that it was a straightforward procedure to perform multiparameter optimisation. It was also possible to establish any optimum value for the control factors within the parameter space, not just at the level settings.

In order to verify the simulation results, wafers were fabricated using the EMF 1.5μm nMOS process. The real measurement results, outlined in **Chapter 8**, showed good agreement with the simulated results, with both indicating a high dose and energy as optimum. The accuracy of the results for both simulation and reality were verified by the different tests used as an indication of the accuracy of the model and of the fit. These tests included the F-test, T-test and the value of the R-squared error. This verified the application of such a software system in a real manufacturing situation, showing the benefits to be obtained using these techniques.

Also compared were two different types of experimental design, D-optimal and full-factorial. Both designs indicated similar optimised conditions for the implant dose and energy. This indicated the potential savings to be made using conventional statistical designs, rather than full-factorial analysis.

One of the benefits of the RSM approach is that once the model has been fitted, no further experimentation is required. This lends itself to integration into a manufacturing environment, to answer "what if" questions. This is particularly useful for devices that have gone out of process specification to see immediately what the ultimate effect will be on particular device characteristics. A possibility for future work would be to integrate such an approach into an expert system type of environment. This would be of great benefit to the ever increasing needs of the semiconductor industry, for instance, on line characterisation information.

In summary, it has been shown that a combination of commercial simulation software, design of experiments and parallel processing can be a useful and efficient tool in the area of IC optimisation and design for manufacturability. It has been illustrated that the use of these tools is essential to reduce the development time for modern integrated circuit designs. The system described offers an efficient and reliable method to perform this optimisation, with a minimum of user interaction. The validity of the system has been verified in the agreement between an optimised design, using simulated and measured data.

Bibliography

[1] T.A.C.M. Classen. Impact of new chip technologies on consumer products. In *Proceedings of the 22nd European Solid State Device Research Conference*, 1992.

[2] P.Chatterjee. ULSI CMOS – the next ten years. In *Proceedings of the 22nd European Solid State Device Research Conference*, 1992.

[3] R.H.Eklund R.H.Havemann. Process integration issues for submicron BiCMOS technology. *Solid State Technology*, pages 71–76, 1992.

[4] The nature of six sigma quality. M.J.Harry, Motorola Inc, Internal report.

[5] Dedicated to total quality. Philips Semiconductors, Promotional literature.

[6] M.W.Jenkins *et al.* The modeling of plasma etching processes using response surface methodology. *Solid State Technology*, pages 175–182, 1986.

[7] G.S.May. Statistical experimental design in plasma etch modeling. *IEEE Transactions on Semiconductor Manufacturing*, 4:83–98, 1991.

[8] G.Z.Yin and D.W.Jillie. Orthogonal design for process optimization and its application in plasma etching. *Solid State Technology*, pages 127–132, 1987.

[9] S.Sharifzadeh, J.R.Koehler, K.B.Owen, and J.D.Shott. Using simulators to model transmitted variability in IC manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 2(3):82–93, 1989.

[10] M.Cecchetti *et al.* Process analysis using RSM and simulation. In *Proceedings of the 22nd European Solid State Device Research Conference*, 1992.

[11] B.E.Deal and A.S.Grove. General relationship for the thermal oxidation of silicon. *Journal of Applied Physics*, 36:377, 1965.

[12] R.W.Dutton D.A.Antoniadis, S.E.Hansen and A.G.Gonzalez. SUPREM I - a program for IC process modeling and simulation. Technical report, Stanford Electronics Laboratories, 1977.

[13] S.E.Hansen D.A.Antoniadis and R.W.Dutton. SUPREM II - a program for IC process modeling and simulation. Technical report, Stanford Electronics Laboratories, 1978.

[14] C.P.Ho, J.D.Plummer, S.E.Hansen, and R.W.Dutton. VLSI process modeling – SUPREM III. *IEEE Transactions on Electron Devices*, 30(11):1438, 1983.

[15] D.Chin, M.R.Kump, H.G.Lee, and R.W.Dutton. Process design using two-dimensional process and device simulators. *IEEE Transactions on Electron Devices*, 29:336, 1982.

[16] SSUPREM IV user's manual, 1990. Silvaco International.

[17] M.R.Pinto, C.S.Rafferty, and R.W.Dutton. PISCES II- user's manual. Technical report, Stanford Electronics Laboratories, 1984.

[18] H.Ryssel *et al.* Simulation of doping processes. *IEEE Transactions on Electron Devices*, 27:1484–1492, 1980.

[19] R.Tielert. Two-dimensional numerical simulation of impurity distribution in VLSI processes. *IEEE Transactions on Electron Devices*, 27:1479–1483, 1980.

[20] J.Lorenz *et al.* COMPOSITE - a complete modelling program of silicon technology. *IEEE Transactions on Electron Devices*, 32(10):1977–1985, 1985.

[21] Titan, 1990. Centre National d'etudies des Telecommunications, Grenoble.

[22] G.E.Smith and A.J.Steckl. A two-dimensional VLSI process modeling program. *IEEE Transactions on Electron Devices*, 29:216–221, 1982.

[23] K.A.Salsburg and H.H.Hansen. FEDSS- finite element diffusion simulation system. *IEEE Transactions on Electron Devices*, 30:1004–1011, 1983.

[24] B.R.Penumalli. A comprehensive two-dimensional VLSI process simulation program: BISCEPS. *IEEE Transactions on Electron Devices*, 30:983, 1983.

[25] Predict users manual, 1991. Silvaco International.

[26] H.K.Gummel. A self-consistent iterative scheme for one-dimensional steady state transistor calculations. *IEEE Transactions on Electron Devices*, 11:455–465, 1964.

[27] S.Selberherr, A.Shultz, and H.W.Potzl. MINIMOS - a two-dimensional MOS transistor analyser. *IEEE Transactions on Electron Devices*, 27(8):1540–1550, 1980.

[28] Z.P.Yu and R.W.Dutton. SEDAN-3 users manual. Technical report, Integrated Circuits Laboratories, Stanford University, 1985.

[29] M.Thurner and S.Selberherr. Three-dimensional effects due to the field oxide in MOS devices analysed with MINIMOS 5. *IEEE Transactions on Computer-Aided Design*, 9:856–867, 1990.

[30] Davinci user's manual, 1990. Technology Modeling Associates.

[31] A.Husain and S.G.Chamberlain. 3D simulation of VLSI MOSFETS: The 3D simulation program WATMOS. *IEEE Transactions on Solid State Circuits*, 17:261–268, 1982.

[32] J.H.Chern *et al.* A 3D device simulator for reliability modeling. *IEEE Transactions on Computer-Aided Design*, 8:516–527, 1989.

[33] S.Selberherr. *Analysis and simulation of semiconductor devices.* Springer-Verlag Wien New York, 1984.

[34] L.W.Nagel and D.O.Peterson. Simulation program with integrated circuit emphasis. *Proc. 16th Midwest Symp. on Circuit Theory*, 1973.

[35] Jau-Yien Lee Shui-Jinn Wang and Chun-Yen Chang. An efficient and reliable approach for semiconductor device parameter extraction. *IEEE Transactions on Computer-Aided Design*, 5:170–179, 1986.

[36] TOPEX user's manual, 1986. Technology Modeling Associates Inc.

[37] S.C.Hu *et al.* Computer-aided design of semiconductor processes and devices. *Philips Journal of Research*, 42:533–565, 1987.

[38] P.Yang and P.Chatterjee. An optimal parameter extraction program for MOSFET models. *IEEE Transactions on Electron Devices*, 30:1214–1219, 1983.

[39] K.Doganis and D.L.Sharfetter. General optimisation and extraction of IC device model parameters. *IEEE Transactions on Electron Devices*, 30:1219–1228, 1983.

[40] R.W.Dutton. Modeling of the silicon integrated-circuit design and manufacturing process. *IEEE Transactions on Electron Devices*, 30:968–986, 1983.

[41] G.M.Koppelman and M.A.Wesley. Oyster: A study of integrated circuits as three-dimensional structures. *IBM J. Res. Development*, 27:149–162, 1983.

[42] K.Lee and A.R.Neureuther. Simpl-2 (simulated profiles from the layout-version 2). *IEEE Transactions on Computer-Aided Design*, 7:160–167, 1988.

[43] M.R.Simpson. Pride: An integrated design environment for semiconductor device simulation. *IEEE Transactions on Computer-Aided Design*, 10:1163–1174, 1991.

[44] J.Mar *et al.* Ease- an application-based CAD system for process design. *IEEE Transactions on Computer-Aided Design*, 6:1032–1038, 1987.

[45] A.J.Strojwas and S.W.Director. The process engineer's workbench. *IEEE J. Solid-State Circuits*, 23:377–386, 1988.

[46] G.E.P.Box, W.G.Hunter, and J.S.Hunter. *Statistics for experimenters : An introduction to design, data analysis and model building.* New York: Wiley, 1978.

[47] C.R.Hicks. *Fundamental concepts in the design of experiments.* Holt, Rinehart and Winston, 1964.

[48] G.E.P.Box and N.R.Draper. *Empirical model-building and response surfaces.* Wiley, 1987.

[49] A.I.Khuri R.H.Myers and W.H.Carter Jr. Response surface methodology. *Technometrics*, 31:137–157, 1989.

[50] BBN Software Corporation. *RS/Discover Statistical Appendices.* BBN Software Products Corporation, 1988.

[51] K.Lin and C.J.Spanos. Statistical equipment modeling for VLSI manufacturing: an application for lpcvd. *IEEE Transactions on Semiconductor Manufacturing*, 3:216–229, 1990.

[52] G.Taguchi. *Introduction to Quality Engineering.* Tokyo:Asian Productivity Organisation, 1986.

[53] G.Taguchi. *Orthogonal Arrays and Linear Graphs.* ASI Press, 1987.

[54] A.P.Turley P.E.Riley and W.J.Malkowski. Development of a multistep SiO_2 plasma etching process in a minibatch reactor using response-surface methodology. *Journal Electrochemical Society*, 136:1112–1146, 1989.

[55] P.E.Riley and D.A.Hanson. Study of etch rate characteristics of SF_6/He plasmas by response-surface methodology: effects of interelectrode spacing. *IEEE Transactions on Semiconductor Manufacturing*, 2:178–182, 1989.

[56] A.R.Alvarez, B.L.Abdi, D.L.Young, H.D.Weed, J.Teplik, and E.R.Herald. Application of statistical design and response surface methods to computer-aided VLSI device design. *IEEE Transactions on Computer-Aided Design*, 7(2):272–287, 1988.

[57] M.S.Phadke *et al.* Off-line quality control in integrated circuit fabrication using experimental design. *The Bell System Technical Journal*, 62:1273–1309, 1983.

[58] Y.Jiannan H.Yie. Progress on model building and statistical analysis methodology of IC characteristics with process. In *IEEE 1991 Conference on Microelectronic Test Structures*, 1991.

[59] D.L.Young *et al.* Application of statistical design and response surface methods to computer-aided VLSI device design II: desirability functions and taguchi methods. *IEEE Transactions on Computer-Aided Design*, 10:103–115, 1991.

[60] R.E.Jones and T.C.Mele. Use of screening and response surface experimental designs for development of a 0.5μm CMOS self-aligned titanium silicide process. *IEEE Transactions on Semiconductor Manufacturing*, 4:281–287, 1991.

[61] RS/1 user's guide, 1988. BBN Software Products Corporation.

[62] RS/Discover user's guide, 1988. BBN Software Products Corporation.

[63] BBN Software Corporation. *RS/Explore Statistical Appendices.* BBN Software Products Corporation, 1988.

[64] Massively parallel computing : Status and prospects, 1992. Edinburgh Parallel Computing Centre report number tr9202.

[65] R.W.Hockney and C.R.Jesshope. *Parallel Computers 2.* Adam Hilger, 1988.

[66] A.J.Anderson. *Multiple Processing A Systems Overview.* Prentice Hall, 1989.

[67] J.P.Hayes. *Computer Architecture and Organisation.* McGraw-Hill, 1989.

[68] A.Trew and G.Wilson. *Past, Present, Parallel.* Springer-Verlang, 1991.

[69] M.Flynn. Some computer organisations and their effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.

[70] R.N.Ibbett. *Architecture of high performance computers volume I.* MacMillan, 1989.

[71] R.N.Ibbett. *Architecture of high performance computers volume II.* MacMillan, 1989.

[72] The Connection Machine CM-200 Series Technical Summary, 1991. Thinking Machines Corporation.

[73] W.D.Hillis. *The Connection Machine.* MIT Press, Cambridge, Massachusetts, 1985.

[74] Cray area of supercomputing, January 1992. What's New in Computing.

[75] C.L.Seitz. The cosmic cube. *Communications of the ACM*, 28, 1985.

[76] Meiko Computing Surface. Meiko Ltd, Bristol.

[77] L.Clarke and G.Wilson. Tiny: An efficient routing harness for the INMOS transputer. *Concurrency: Practice and Experience*, 3:221–245, 1991.

[78] CStools for Meikos, April 1991. Meiko Ltd.

[79] N.Carriero S.Ahuja and D.Gelernter. Linda and friends. *Computer*, 19:26–34, 1986.

[80] I.Foster and S.Taylor. *Strand - New concepts in parallel programming.* Prentice-Hall, 1990.

[81] R.Allan. Distributing the load. *Parallelogram*, August:18–19, 1991.

[82] The helios operating system, 1989. Perihelion Software Ltd.

[83] An introduction to occam 2 programming, 1988. Physics Dept, University of Edinburgh.

[84] W.J.C.Alexander *et al.* The implementation of process simulation on transputers for the production of ASICs. In *8th Australian Conference on Microelectronics*, 1989.

[85] INMOS Ltd. *The Transputer Databook.* Bath Press Ltd, 1988.

[86] M.Homewood. The IMS t800 transputer. *IEEE Micro*, pages 10–26, October 1987.

[87] INMOS Ltd. *Occam 2 Reference Manual.* Prentice Hall, 1988.

[88] 3L Toolset. 3L Ltd, Livingston, Scotland.

[89] IDRIS Version 5, Oct 1991. Parsys Ltd.

[90] R.Dettmer. Great expectations. *IEE Review*, pages 411–413, December 1990.

[91] T.S.Perry. Intel's secret is out. *IEEE Spectrum*, pages 22–28, April 1989.

[92] E.Henning. Intel's RISC revolution for the '90s. *PC User*, 29:40–52, 1989.

[93] Transtech times, 1992. Transtech Parallel Systems Ltd, High Wycombe.

[94] Computing Surface Overview, 1991. Meiko Ltd, Bristol.

[95] K.M.Chandy and C.Kesselman. Parallel programming in 2001. *IEEE Software*, pages 11–20, November 1991.

[96] Beyond the Supercomputer, Parsytec GC, 1991. Parsytec Ltd.

[97] D.D'Arcy. Alpha chip gives rivals food for thought. *Electronics Weekly*, page 9, March 1992.

[98] D.D'Arcy. Supercomputing in parallel. *Electronics Weekly*, page 10, March 1992.

[99] G.J.Gaston *et al.* IC optimisation using RSM and parallel processing. In *Applications of transputers 3, IOS Press*, 1991.

[100] G.J.Gaston *et al.* A general taskfarming tool: Its application to semiconductor fabrication. In *Parallel Computing: From theory to sound practice*, 1992.

[101] Occam toolset, user manual, 1988. INMOS Limited.

[102] CSTOOLS Communications Library, 1990. Meiko Ltd, Bristol.

[103] A tutorial introduction to CSTOOLS, 1990. Meiko Ltd, Bristol.

[104] Mr Z. Chen, private communication.

[105] K.Shimohigashi J.J.Barnes and R.W.Dutton. Short-channel MOSFET's in the punchthrough current mode. *IEEE Transactions an Electron Devices*, 26:446–452, 1979.

[106] T.Nakamura *et al.* Sub-micron channel MOSFET's logic under punchthrough. *IEEE Journal Solid State Circuits*, 13:572, 1978.

[107] C.Hu F.Hsu, R.S.Muller and P.Ko. A simple punchthrough model for short channel MOSFET's. *IEEE Transactions on Electron Devices*, 30:1354–1359, 1983.

[108] Y.P.Tsividis. *Operation and modelling of the MOS transistor.* McGraw-Hill, 1988.

[109] R.R.Troutman. VLSI limitations from drain-induced barrier lowering. *IEEE Transactions on Electron Devices*, 26:461–468, 1979.

[110] B. El-Kareh and R.J. Bombard. *Introduction to VLSI silicon devices.* Kluwer Academic Publishers, 1986.

[111] E.S.Yang. *Microelectronic devices.* McGraw-Hill, 1988.

[112] S.M.Sze. *Physics of Semiconductor Devices.* Wiley and Sons, 1981.

[113] R.Ford. The application of Taguchi methods to the optimisation of IC fabrication processes, May 1991. B.Eng Hons Project Report HSP794.

[114] M.Davidson, K.Kaufman, and I.Mazor. First results of a product utilizing coherence probe imaging for wafer inspection. In *SPIE Integrated Circuit metrology, Inspection and Process Control*, volume 921. SPIE, Feb-Mar 1988.

Appendix 1

Reprints of two published papers relevant to this thesis.

IC Optimisation using RSM and Parallel Processing

G.J. Gaston, A.J. Walton and W.K.Choi.

Edinburgh Microfabrication Facility, Department of Electrical Engineering, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

Abstract

This paper details how a combination of statistical methods, parallel processing and simulation tools can be used in the optimisation of Integrated Circuit (IC) processes. Response Surface Methods (RSM) and simulation programs are described, together with different control factors and output responses. A software system, utilising an array of transputers is outlined and an example response surface plot given.

1. Introduction

As a result of the trends towards greater complexity and miniaturisation of VLSI devices, modern IC's are becoming increasingly susceptible to fluctuations in the manufacturing process. It is such complexities that have heralded the use of Computer-Aided Design (CAD) tools for process and device simulation. From quiet beginnings these tools are now widely used in the semiconductor industry.

The major advantage of these simulation programs is that costly prototype production runs, which can take months to complete, can be replaced by simulated experiments. These simulations can be completed in a matter of hours or days, depending on the processing capability of the computer and the type of simulation involved. As a result the length of the product development cycle is kept to a minimum, which is a critical aim of all semiconductor maufacturers.

Statistical methods, known as design centering or Response Surface Methodology, are used to reduce the number of simulations required, in order to

arrive at optimum settings for the manufacturing equipment [1,2]. Thus a set of control factors are varied and their effect on some output responses are analysed. Using RS/1 [1] a set of experiments can be designed, containing different settings for each control factor. There may be upwards of 50 separate simulations that need to be carried out to determine the optimum setting, depending on the number of factors to be varied.

Although simulations reduce the time greatly, when compared to prototype runs, they can still be very CPU intensive and each simulation may take hours to complete. Since there are say, 50 different simulations to be executed, this lends itself greatly to a parallel taskfarm approach on transputers. Each transputer, called a slave or worker, in an array receives a task from a master transputer or driver and on completion the slave then receives the next job information. The complete procedure has been automated in a software system, thus keeping human interaction to a minimum.

2. Simulation

2.1. Process Simulation

As a result of the need to reduce the costs of prototype production runs, process simulation has become an essential tool in the process engineer's tool kit. Process simulation is particularly important in the evaluation of new structures or in the improvement of existing structures. These improvements may include changes for increased manufacturability.

Such programs can be divided into 1-D and 2-D process models. The 1-D programs such as SUPREM3 [3], have very sophisticated models for the different procedures in the manufacturing process. Accurate results can thus be obtained, but only a 1-D section through the structure can be simulated.

The 2-D programs such as SUPREM4 [4], SUPRA [5], are essential for analysing small geometry processes, where 2-D effects become more important. These simulations are, however, more CPU intensive.

The structure and doping profile calculated by process simulation, provide essential information required by device simulation, in order that accurate device characteristics can be simulated.

[1] A statistical package from BBN Software

2.2. Device Simulation

The results of the device simulator are obviously profoundly affected by the input data generated by the process simulator. As the name suggests, device simulation is concerned with the device characteristics of the final product, eg I-V curves. Essentially, device simulation involves the solving of the basic semiconductor equations, which govern the static and dynamic behaviour of carriers in semiconductors under the influence of external fields [6]. There are a large number of device simulators available at present [7-9].

The output of the device simulator, in the form of I-V curves, enables circuit parameters to be extracted for use with circuit simulation programs. Such parameters include, threshold voltage, series resistance and transconductance.

3. Response Surface Methodology And Experimental Design

Response Surface Methodology (RSM) techniques have been employed in order to reduce the number of simulations that are required to give the necessary information about a particular process.

As Box states, these techniques are essentially addressing the following questions [1]:

- How is a particular response affected by a given set of input variables over some specified region of interest?
- What settings, if any, of the inputs will give a product simultaneously satisfying desired specifications?
- What values of the inputs will yield a maximum for a specific response, and what is the response surface like close to this maximum?

The program RS/1, is used for setting up the experimental design. It enables the user not only to specify the required factors and responses to investigate, but also the ability to define a number of different attributes for each factor, eg tolerance, settings, precision. It is also possible to model several responses in one experiment.

Depending on the choice of design, it is possible to greatly reduce the number of simulations that need to be executed. The design types that can be used in RS/1 include Full-Factorial, Box-Behnken, D-Optimal and Central Composite design. Utilising RS/1 one can, therefore, use a planned program

of experimentation (simulation) to obtain a complete response surface for all combinations of input factors from a reduced number of runs. An unstructured approach will not only be significantly heavier on CPU time, but also will not necessarily point to the optimum processing conditions.

RSM can be used to fit response surfaces, to analyse and interpret the results obtained from the simulations. The data may be visualised using graphical techniques, analysis of variance performed and models fitted to the results. Routines can be written using RPL, the RS/1 programming language, to enable quick and efficient transfer of files, tables etc. The RPL can thus be used as an interface between RS/1 and the simulation results.

Table 1 shows part of a worksheet containing different control factors and responses for a typical example of a bipolar process. The control factors of interest are oxide thickness, boron dose, boron energy, anneal temperature and arsenic dose.

4. Software System

In order to simulate the different runs it is necessary to create as many run files for the simulator as there are experiments. A program has been written to automatically read the control factor settings output from RS/1 and generate the required drive files. This program reads a control file which defines filenames, responses, etc. This results in an extremely flexible approach to generate files with a large diversity of combinations and permutations. Fig 1 shows the structure of the software system.

The data generation program reads the control file and from the generic file and experimental settings, generates the necessary simulation input files.

These different drive (input) files can then be farmed out onto an array of transputers using PARTICS (paper to be published) with each transputer running an identical copy of the process simulator. PARTICS operates on the principle of a master transputer allocating tasks to an array of slave transputers. It provides the facility to rename files, redirect stdin and stdout, and assign different names to filenames generated during the simulation. This is achieved by placing a "shell" around the executable code, thus no changes are required to the source code. All that is required is a compiled version of the user program on a transputer and the system can be used.

Two different platforms can be used to perform the simulations. The code was originally written for the IBM PC or compatible, with a rack of T800 and

0	1 OX_THICK (um)	2 BORON_ENERGY (kev)	3 BORON_DOSE (atoms/cm2)	4 ARSENIC_DOSE (atoms/cm2)	5 TEMP (oc)
1	0.01	20.00	9e+13	3e+15	1150.00
2	0.09	20.00	9e+13	3e+15	1000.00
3	0.01	35.00	9e+13	3e+15	1000.00
4	0.09	35.00	9e+13	3e+15	1150.00
5	0.01	20.00	2e+14	3e+15	1000.00
6	0.09	20.00	2e+14	3e+15	1150.00
7	0.01	35.00	2e+14	3e+15	1150.00
8	0.09	35.00	2e+14	3e+15	1000.00
9	0.01	20.00	9e+13	1e+16	1000.00
10	0.09	20.00	9e+13	1e+16	1150.00
11	0.01	35.00	9e+13	1e+16	1150.00
12	0.09	35.00	9e+13	1e+16	1000.00
13	0.01	20.00	2e+14	1e+16	1150.00
14	0.09	20.00	2e+14	1e+16	1000.00
15	0.01	35.00	2e+14	1e+16	1000.00
16	0.09	35.00	2e+14	1e+16	1150.00
17	0.01	27.50	1.45e+14	6.5e+15	1075.00
18	0.09	27.50	1.45e+14	6.5e+15	1075.00
19	0.05	20.00	1.45e+14	6.5e+15	1075.00
20	0.05	35.00	1.45e+14	6.5e+15	1075.00
21	0.05	27.50	9e+13	6.5e+15	1075.00
22	0.05	27.50	1.45e+14	6.5e+15	1075.00
23	0.05	27.50	1.45e+14	3e+15	1075.00
24	0.05	27.50	1.45e+14	1e+16	1075.00
25	0.05	27.50	1.45e+14	6.5e+15	1000.00

Table 1: Experiment Worksheet

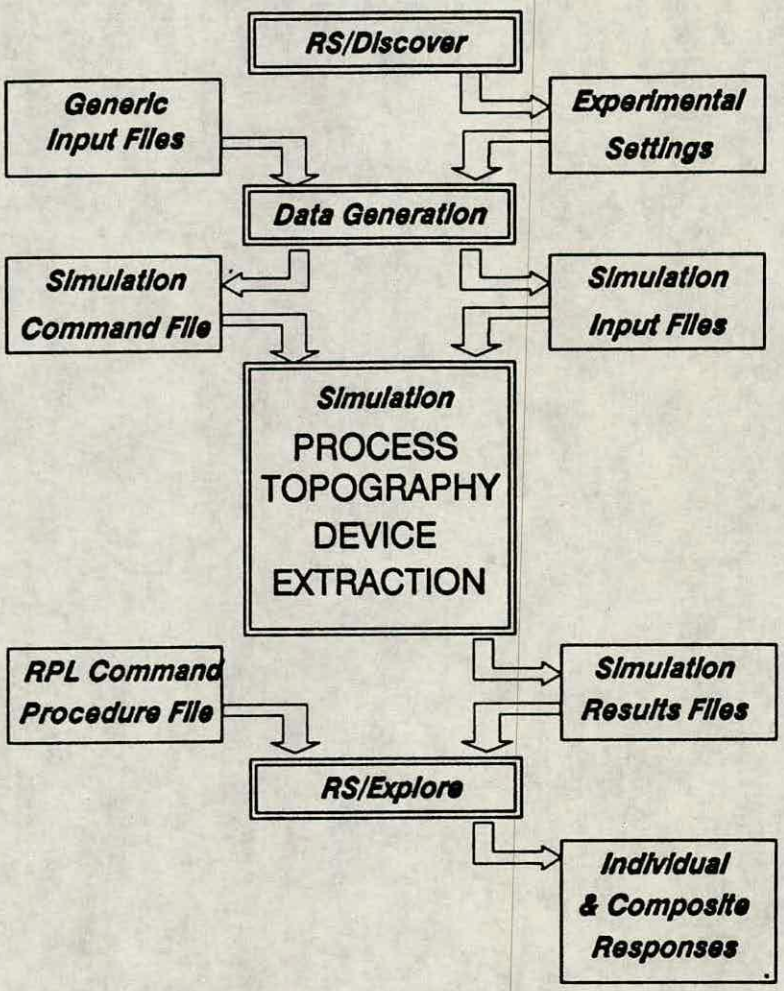


Figure 1: Complete Software System

T400 transputers. This was later ported onto the MEiKO Computing Surface at the University of Ediinburgh, containing over 400 T800 transputers. Fig 2 outlines the operation of the taskfarm on the PC. Tiny is a message passing system written by the Edinburgh Parallel Computing Centre. Messages are passed to the PC from the worker by converting from an AFS protocol to a Tiny protocol and then back to an AFS protocol. As a result each worker transputer has a pseudo access to the host filing system, for opening and closing of files etc.

The profiles from the process simulations can then be automatically input into a device simulator and these simulations farmed out onto the array. The results from the device simulations can then be loaded into RS/1 for statistical analysis and the response surfaces plotted. Thus for the given example of the bipolar process, contourplots can be output showing the variation of gain (beta) and base width (B_W) with the specified control factors, as shown in fig 3. This plot also includes a composite response, C, from which it is possible to reach an optimum setting for the critical manufacturing processes.

5. Conclusions

The system offers a very fast and efficient method for taskfarming a number of CPU intensive programs. The automation of the procedure, in addition to speeding up the process, has also largely eliminated the possibility of human error which often occurs due to the large number of files which require editing. The major attraction, however, of the above approach is that a systematic evaluation of the fabrication process can be performed in little more than the time normally taken for a single run.

The use of PARTICS has a very significant improvement on the elapsed time to optimise a process, reducing a week of CPU time to that of only a few hours. This can be achieved using a supercomputer like the MEiKO Computing Surface. A PC system can also be used to reduce the execution time, thereby reducing computing costs.

7. Acknowledgments

The authors would like to thank the Dept. of Education N. Ireland for the funding of the project, and the Edinburgh Parallel Computing Centre for the provision of the message passing system, TINY.

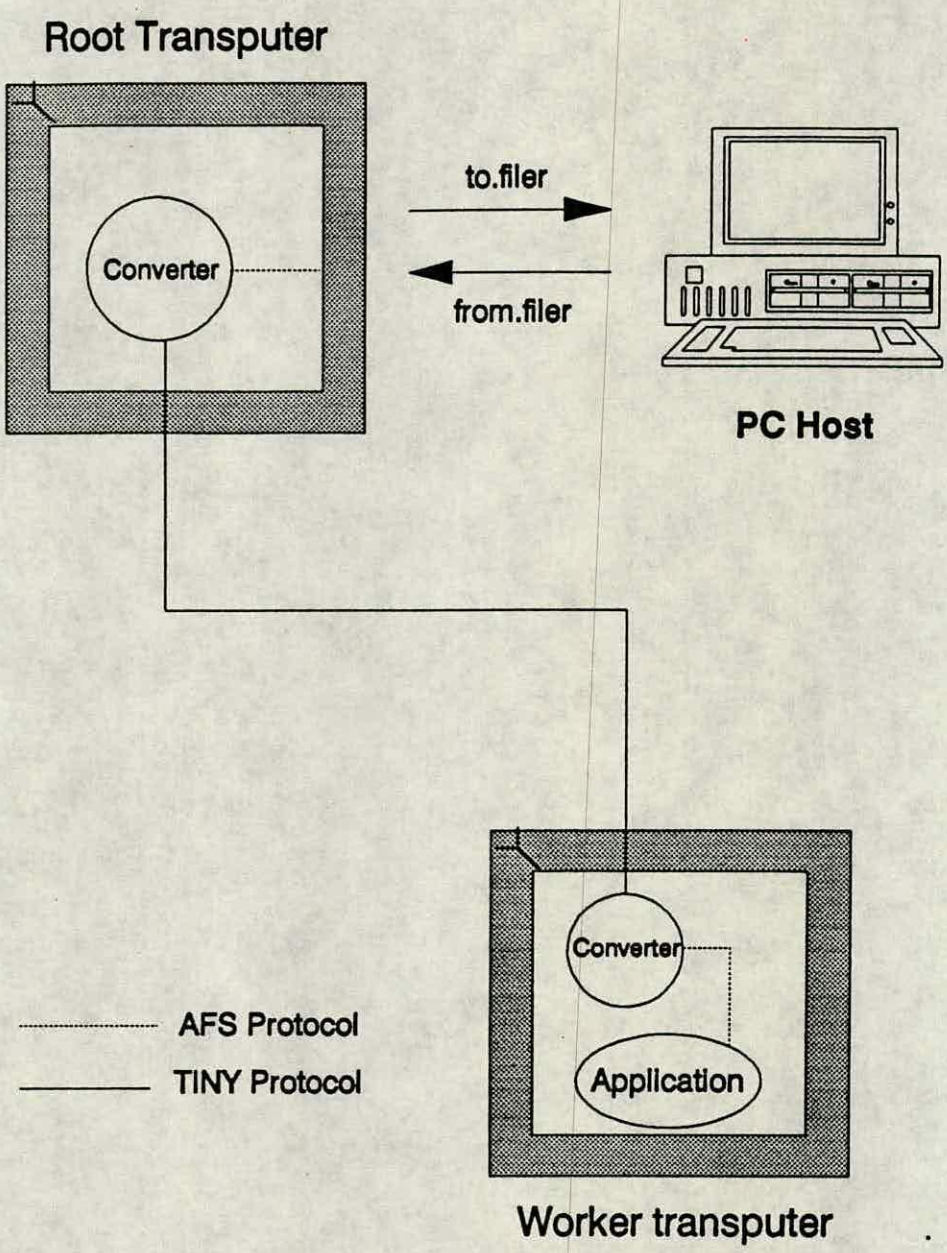


Figure 2: Taskfarm Operation

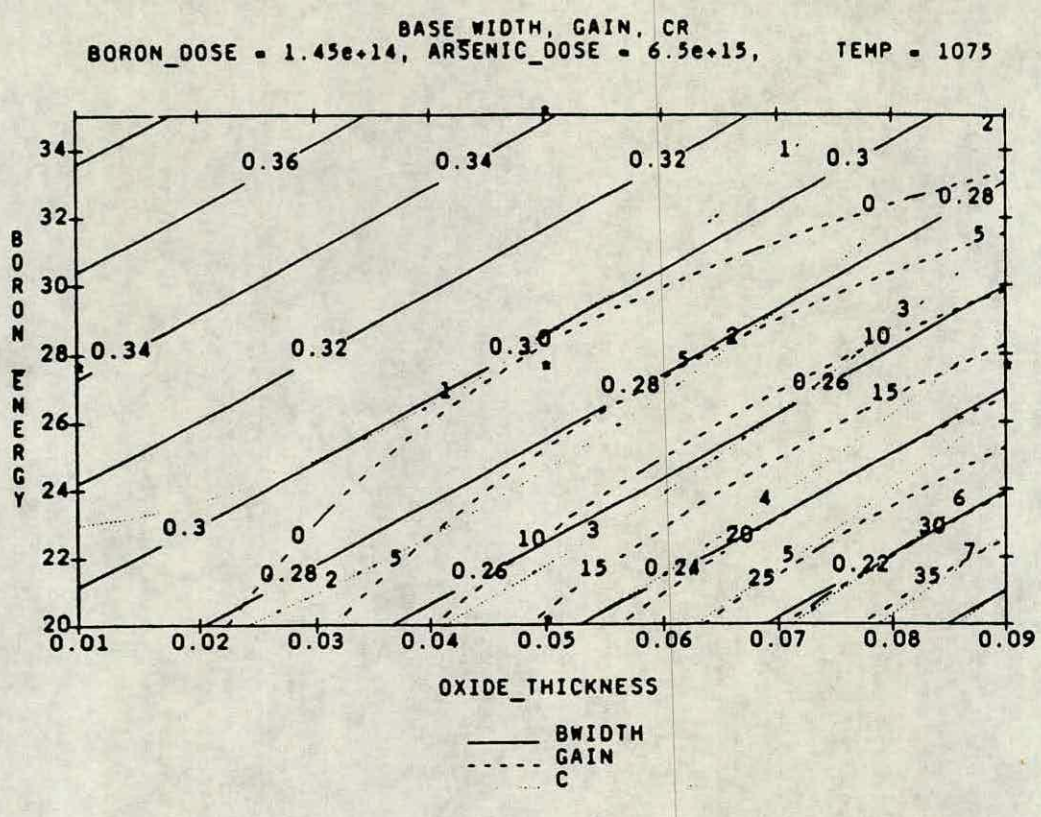


Figure 3: Contourplot

8. References

1. G.E.P.Box and W.G.Hunter and J.S.Hunter. *Statistics for experimenters : An introduction to design, data analysis and model building* New York: Wiley 1978.

2. A.R.Alvarez et al. Application of statistical design and response surface methods to computer-aided VLSI device design. *IEEE Transactions on Computer Aided Design* 7(2): 272-278, 1988.

3. C.P.Ho and J.D.Plummer and S.E.Hansen and R.W.Dutton. VLSI Process modeling – SUPREM III. *IEEE Transactions on Electron Devices* 30(11):1438 1983.

4. SUPREM4 - Stanford University.

5. SUPRA - Stanford University.

6. S.Selberherr *Analysis and Simulation of Semiconductor Devices* Springer-Verlag Wien New York, 1984.

7. PISCES - Stanford University.

8. S.Selberherr and A.Shultz and H.W.Potzl. MINIMOS - A two-dimensional MOS transistor analyser *IEEE Transactions on Electron Devices* 27(8):1540-1550, 1980.

9. Z.P.Yu and R.W.Dutton. SEDAN-3 User's Manual 1985.

A General Taskfarming Tool : Its Application to Semiconductor Fabrication

G.J. Gaston, W.J.C. Alexander, L.J. Clarke † and A.J. Walton.

Edinburgh Microfabrication Facility, Department of Electrical Engineering, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

† Edinburgh Parallel Computing Centre, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

ABSTRACT

A universal tool that can be utilised to farm cpu intensive tasks onto an array of transputers is outlined. The operation of the program is described together with its versatility and functionality. The results obtained using the tool in conjunction with the semiconductor fabrication simulator, SSUPREM4 are detailed. The results show that for very little effort a considerable speedup can be obtained when compared to coventional sequential programming techniques.
Keywords: Taskfarming, transputers, parallel processing, simulation.

1. INTRODUCTION

Semiconductor fabrication technology has now increased to the level of several million transistors on a single silicon chip. Original methods of experimentation by trial and error to create a new process or to optimise an existing one are no longer financially viable. This is due to the ever increasing costs of development both in time and resources. Prototype runs can cost tens of thousands of dollars and may take weeks or months to complete. To alleviate these problems, the major semiconductor fabrication companies are now using simulation software to establish processes and to reduce the number of real experimental evaluations that would otherwise be required. Although simulations reduce the time greatly, when compared to prototype runs, they can still be very CPU intensive and each simulation may take hours or days to complete, depending on the processing capability of the computer and the type of simulation involved. As a result the length of

the product development cycle is kept to a minimum, a critical aim of all semiconductor maufacturers.

The whole technique of experimentation necessitates performing different evaluations on the same process but with different process variables. For example it may be required to see the effect on threshold voltage, by changing the diffusion time and temperature for the gate oxidation step and the value of dose for an implant step. This would result in a large number of experiments that would need to be carried out, with 20 or 30 not being uncommon. Since each experiment (or simulation) is totally separate from the others this lends itself to a taskfarm arrangement on an array of transputers. Each transputer executes a sequential copy of the simulation code but with a different data set. It should be noted that statistical techniques are available where the total number of simulations can be reduced for a given number of variables [1].

It is the purpose of this paper to outline a taskfarming environment on an array of transputers at the Edinburgh Parallel Computing Centre. The computer used is a MEiKO Computing Surface, containing over 400 T800 transputers. The taskfarm operates in the usual mode of a master transputer allocating tasks to a number of worker transputers.

2. GENERAL OUTLINE OF TASKFARM

2.1. Taskfarm Requirements

The following is an outline of the salient features that were required in the design and construction of the taskfarm system:

- There should be no changes to the user source code to facilitate the operation of the system.
- The system should be flexible to use with any user software.
- One file should control all user interaction.

One of the major problems of parallel processing at present is the large investment in time and money that is often required to change the operation of sequential code to that on a parallel computer eg. an array of transputers. This usually means at best, making changes to the source code or at worst rewriting the code in another language eg. occam. In the academic world

availability of the user source code is usually not a problem and given the necessary time and effort the code can be sufficiently changed to run on a taskfarm environment. In a commercial situation, however, with code often costing hundreds of thousands of dollars, or with commercial secrecy a problem concerning the operation of software, source code availability is not guaranteed. As a result it is essential that commercial code should be able to run on the taskfarm and all that would be required would be recompilation of existing code on the transputers, something that could easily be carried out by the company itself.

For reasons already discussed it was also considered important to be able to have a general taskfarm system that could be easily adapted to be used with any user software. This would again reduce the time and effort required in porting code onto parallel machines.

By providing the facility of a control file, any changes that are then required to the system can be implemented by a minor editing change of a single file.

2.2. Problems to Overcome

Given the strict guidelines outlined in the previous section, many problems presented themselves. The following section describes the major problems to be overcome. Section 3 explains how these complications were surmounted.

1. Due to an identical copy of the code running on each worker transputer, all workers would try to read from the keyboard, write to the screen and read/write from/to the same files. It was essential that any solution would remove such a scenario.

2. Since each worker thinks that it is the only code running and that it alone has complete access to the filing system, all workers must be given a pseudo access to the filing system. This will require some form of communication system to pass file requests to and from the filing system from each worker.

3. For large jobs ie. with a large number of worker transputers, there may be a limit to the number of files that can be kept open at any one time. For example if the maximum limit of the system in question is

32 files and there are 16 workers this means that only two files can be open at any one time on each worker. For a PC system the maximum value that can be opened is only 15. It is important that this limitation should be allowed for in the taskfarm.

2.3. Simulation Code

For the purposes of this taskfarm the semiconductor fabrication simulation software used was called SSUPREM4 [2]. This particular package comes under the generic name of a process simulator. As the name suggests it is used to determine the structure and concentration levels of a semiconductor device and how these parameters change with different process parameters. Such programs are used to provide detailed information to device simulators which analyse the electrical characteristics.

SSUPREM4 is one of the most advanced and accurate of all the different process simulation packages that are available on the market at present. As a result of this it is also the most computationally intensive and thus the most suitable to realisation on a network of transputers. The program operates by reading in an input data file and then modelling the different processes detailed sequentially in the input file.

3. OPERATION OF TASKFARM

3.1. Jobfile

In order to overcome the problem of each worker reading the keyboard and writing to the screen etc. it was necessary to define a convention for the renaming of files. This lead to the formation of a jobfile which details the renaming convention eg. redirection of stdin, stdout and any other files peculiar to the user code. Such a file for the operation of SSUPREM4 is outlined below.

```
begin BLOCK-1
do sup4.b4
assign SSUP4USKEY SSUP4USKEY1
assign SSUP4MOD SSUP4MOD1
assign SSUP4IMP SSUP4IMP1
```

```
stdin run01.in
stdout run01.out
done

do sup4.b4
assign SSUP4USKEY SSUP4USKEY2
assign SSUP4MOD SSUP4MOD2
assign SSUP4IMP SSUP4IMP2
stdin run02.in
stdout run02.out
done
.
.
.
end
```

This specifies that worker transputer number 1 will receive input from run01.in, print output to run01.out (rather than the screen) and assign any other files created by SSUPREM4 to a unique name eg. SSUP4USKEY to SSUP4USKEY1 etc. The jobfile is specified in blocks, the next block will not be started until the previous block is finished. This adds an element of control over the order or preference in which the simulations are executed and thid is of importance if a device simulation is to follow a process simulation.

3.2. TINY

In order to be able to open and assign different names to files, it is essential that each worker transputer has access to the filing system. To facilitate this, an efficient message passing is used to transfer messages from the worker to the master transputer. The system used was called TINY [3], a product of the Edinburgh Parallel Computing Centre. TINY can be used to determine processor topology, process mapping, calculate the routing tables and read and write messages. It has a specific protocol which is used to send and receive messages. To send a message the message type must be specified, along with the destination, and the name and size of the buffer to be sent. This protocol is outlined below:
To send data via TINY the buffer, Buf, is loaded and sent using `t_sseq`

eg. `t_sseq(AFS_TYPE,dest,Buf,sizeof(Buf))`. A similar convention is used to receive data except the routine `t_recv` is called.

3.3. Fileserver Protocol

SSUPREM4, which is written in C, was compiled on the transputers using 3L Parallel C [4]. Using 3L utilities, namely the afserver, (which is part of the Parallel C package) SSUPREM4 will communicate with the filing system at a low level, using a fileserver protocol. In order to pass messages around the network it is essential to use the TINY protocol. The file system expects to receive information in the following file protocol format [5]. For example a 32 bit integer is sent as `int32.value;INT32`, where `int32.value` indicates that a 32 bit integer is about to be sent and `INT32` contains the actual value. A record is sent as `record32.value;INT32::[]BYTE`, meaning a record is about to be sent, followed by the size of the record, followed by the record itself.

integer is defined as `int 32.value; INT32`

record is defined as `nilrecord.value` or `record32.value;INT32::[]BYTE`

This can best be illustrated by taking an example of the OpenFile command. As the name suggests this is sent to the filing system from the worker transputer each time a file needs to be opened in the simulation program. Figure 1 outlines the route this command must take to reach the filing system. The SSUPREM4 program requests to open a file. The filer protocol, as outlined in [5], needs to send the following parameters to the filing system: openfile.cmd, filename, access.method, open.mode, exist.mode and record.length. These represent a variety of `int32.values` and `record32.values`. This information is, therefore, stored in a buffer and sent down the TINY message passing system. This effectively changes the fileserver or AFS protocol to the TINY protocol.

At the master transputer the message is received and converted back into the fileserver protocol and the necessary file opened at the filing system. An acknowledgement is then sent back to the worker, with details of the ID number associated with that particular file, again changing from fileserver protocol to TINY protocol and back.

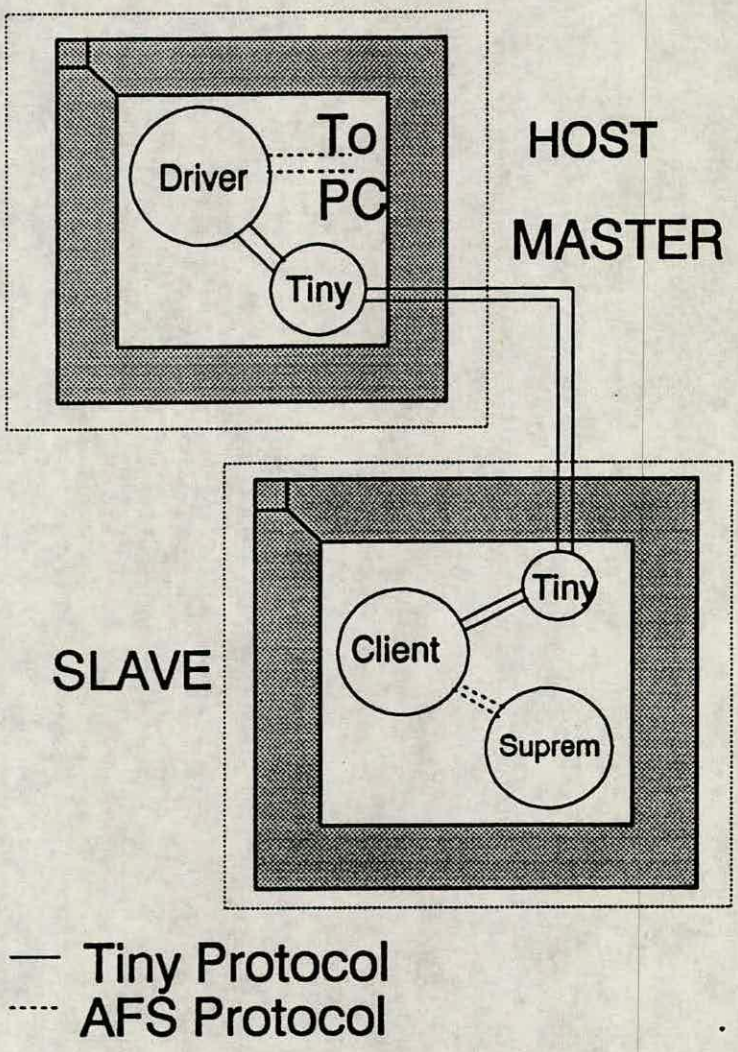


Figure 1: Outline of protocols and message transfer

3.4. General Structure of Program

As can be seen from figure 1 there are two main parts to the taskfarm:

1. Driver
2. Client

Driver The following is a brief description of the driver program, which is written in 3L Parallel C.

- Receives TINY protocol commands from worker
- Converts to AFS Protocol
- Executes AFS function eg. openfile command
- Sends TINY protocol acknowledge to worker

The driver is used to coordinate the allocation of tasks, collection of results, reading of the jobfile, etc. The driver program is also responsible for checking the maximum number of open files allowable on the system and comparing this with the number actually open at present. If, on receipt of an openfile command, the system number will be exceeded then it is the function of the driver to close a file to facilitate the opening of the new file. A priority is given to each file and thus low priority files will always be closed before a high priority file.

Client The following is a brief description of the client program, which is also written in 3L Parallel C.

- Receives AFS protocol commands from SSUPREM4
- Makes any necessary changes to the filename
- Converts to TINY Protocol
- Sends to driver
- Receives acknowledge from driver

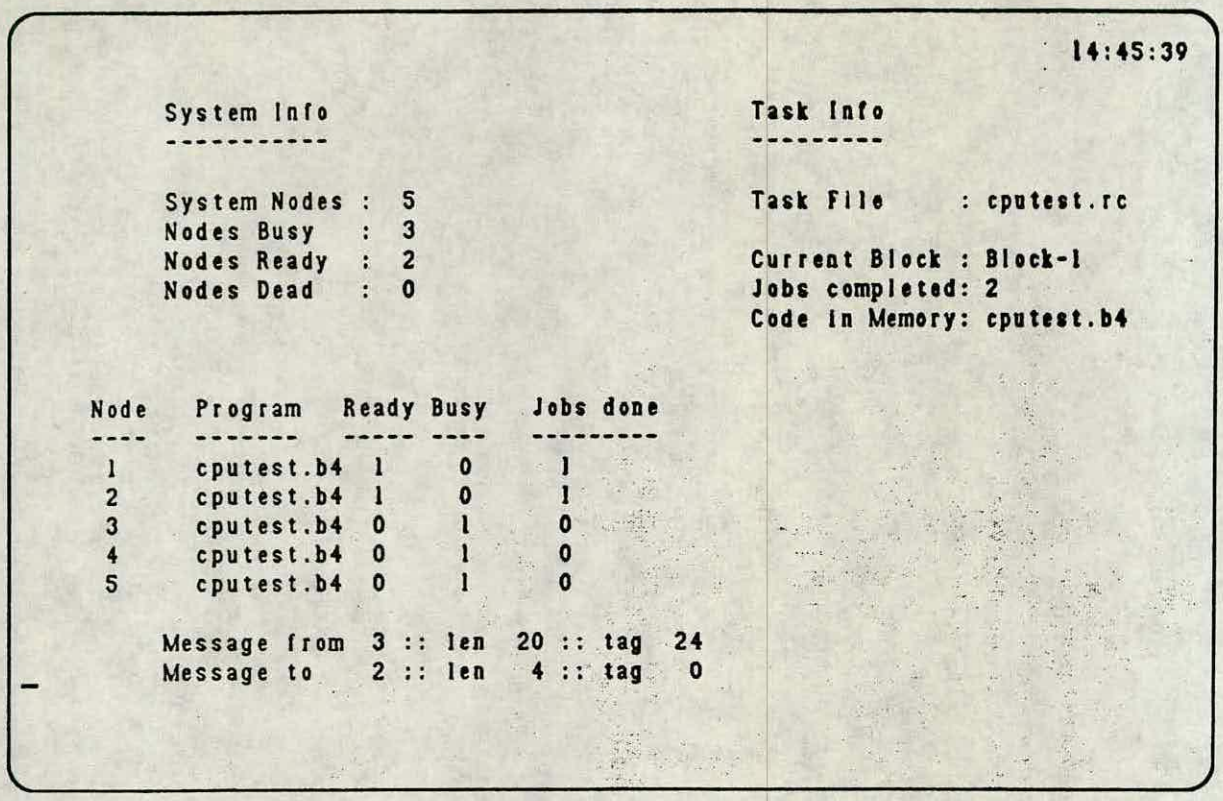


Figure 2: System screen information

The client is responsible for, among other tasks, translation of filenames as outlined in the jobfile. For example on request of an openfile for S-SUP4USKEY on worker number 2, the client will translate this to an openfile for SSUP4USKEY2, and this information sent on to the driver as normal.

Figure 2 shows the information provided by the system at runtime. The 5 nodes refer to 5 worker transputers on the system. For the sake of clarity only 5 workers were used but this number can be increased to the maximum number of transputers available. In order to speed up the loading of large executable codes, they are loaded from memory, if they are already resident. This speeds up the total execution time as disk access is much slower.

4. RESULTS

Figure 3 shows the results obtained when using the taskfarm with S-SUPREM4 and different data files. The results are for a farm of 10 transputers and the time for a SUN4 is the total time taken to run 10 sequential jobs. As the task number increases, so does the computational requirements of

the job. As a result any communications overhead of the transputer system becomes less and less significant as the cpu times increase. This is evident by comparing the elapsed times for task 1 and task 4 for the two different systems.

For the case of SSUPREM4, there are three different coefficient files to be loaded by each worker, thus making a sizeable contribution to the communications time. For an application that uses less loading of files then the speedup would be even more spectacular. Obviously if more transputers were used then the speedup, when compared to sequential techniques, would be even greater and almost linear in nature. It is worth noting that for this application the SUN4 is about four times faster than the transputer, when both are used in serial mode. This illustrates with even more clarity the benefits of the taskfarm system.

The above results were only obtained using relatively small simulations and if a complete process was to be simulated then the cpu time is likely to be in hours or days rather than minutes. Thus the speedup of the taskfarm system would be even more beneficial.

5. CONCLUSIONS

In conclusion the taskfarm offers almost linear speed up for each transputer added to the system configuration. It is not essential to change the source code of the application software and the "shell" allows redirection of the necessary files. The potential commercial value to this is obvious as no inherent knowledge about parallel computing is required, just the ability to recompile existing code. The jobfile concept allows ease and flexibility, resulting in different simulation code running concurrently on different transputers if required. To use the system with a particular application, all that is required is a minor editing change to the jobfile and the system can be utilised with any user program.

The use of the taskfarm has a very significant improvement on the elapsed time to simulate a process, reducing a week of CPU time to that of only a few hours. This can be achieved using a supercomputer like the MEiKO Computing Surface. This system has also been implemented on a PC system with very similar performance and at a system cost that is affordable for small companies.

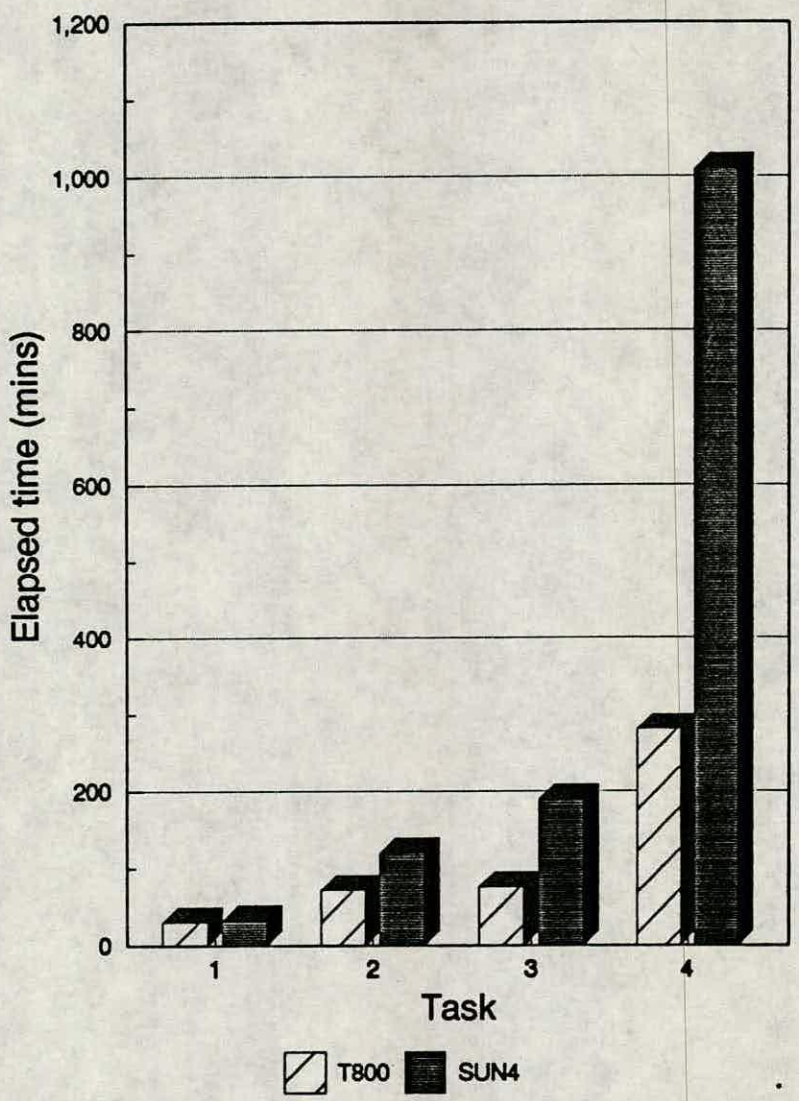


Figure 3: Comparison of taskfarm and SUN4

7. ACKNOWLEDGMENTS

The authors would like to thank the Dept.of Education N.Ireland for the funding of the project, INMOS (Bristol) for their help and support and the Edinburgh Parallel Computing Centre for the provision of the message passing system, TINY. We would also specially thank Silvaco Data Systems for the provision of the simulator SSUPREM4.

8. REFERENCES

1. G.J.Gaston, A.J.Walton and W.K.Choi. IC Optimisation using RSM and Parallel Processing *Applications of Transputers 3 1991 p203.*
2. SSUPREM4 Silvaco Data Systems 1991.
3. L.J.Clarke, Ph D. Thesis, University of Edinburgh 1990.
4. 3L Parallel C Manual V2.1.
5. INMOS Limited. *Occam Toolset User Manual* 1988.